Verification of Incomplete Graph Unlearning through Adversarial Perturbations

Kun Wu
kwu14@stevens.edu
Stevens Institute of Technology
Hoboken, New Jersey, USA

Wendy Hui Wang
hwang4@stevens.edu
Stevens Institute of Technology
Hoboken, New Jersey, USA

ABSTRACT

Graph unlearning (GU) enables data owners to remove specific data from a trained Graph Neural Network (GNN). However, a dishonest model provider may cheat on the unlearning process. This paper focuses on a specific type of cheating behavior in GU, namely *incomplete edge unlearning*, where the model provider removes only a subset of the requested edges from the trained GNN. We introduce *PANDA*, the first probabilistic GU verification framework, to detect such cheating behaviors. PANDA identifies a set of nodes, called *token nodes*, in the graph, and injects a set of fake edges, referred to as *challenge edges*, into the model to manipulate the predictions of token nodes. A key property of the challenge edges is that removing only a subset of them from the poisoned model does not change the token nodes' prediction. Then by requesting the removal of the challenge edges and observing the change in the token nodes' predictions, the verifier can assess the likelihood that the model provider has engaged in incomplete unlearning. To develop PANDA, we design a novel algorithm to identify the token nodes and generate their associated challenge edges. We rigorously quantify the verification probabilities achieved by PANDA. Our extensive empirical studies demonstrate the efficiency and effectiveness of PANDA in detecting incomplete edge unlearning across across a variety of GNN models and unlearning algorithms. Furthermore, we show that PANDA exhibits strong robustness against state-of-the-art detection methods for graph adversarial perturbations. Our code and datasets are available at https://github.com/kunwu522/unlearning-verification-gnn.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → *Machine learning*;

KEYWORDS

Integrity Verification of Graph Unlearning; Machine Unlearning

ACM Reference Format:
Kun Wu and Wendy Hui Wang. 2025. Verification of Incomplete Graph Unlearning through Adversarial Perturbations. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3711896.3737179

KDD '25, August 3–7, 2025, Toronto, ON, Canada

ACM ISBN 979-8-4007-1454-2/2025/08…$15.00
https://doi.org/10.1145/3711896.3737179

1 INTRODUCTION

Graph-structured data has become a rich source of insights in a broad spectrum of application domains such as social systems, biological networks, and information systems. Graph Neural Networks (GNNs) [23] have emerged as powerful tools for analyzing graph-structured data [40, 49].

Despite their success, deployments of GNNs raise privacy concerns when the input graphs contain sensitive information such as users' sensitive social relationships and health information. To protect users' information, several regulations such as the General Data Protection Regulation (GDPR) [20] and the California Consumer Privacy Act (CCPA) [19] mandate that organizations (model owners) uphold the "right to be forgotten" (RTBF) [17]. By RTBF, the users (data owners) can request the removal of their information from the trained models.

To address RTBF for GNNs, several efficient graph unlearning (GU) algorithms have been developed recently [6–8, 18, 29, 33, 39, 48, 50]. Despite their effectiveness, these techniques are not transparent for data owners, i.e., data owners cannot monitor the unlearning process and verify whether their data has been truly removed from the model [47**?**]. For instance, under the Machine-Learning-as-a-Service (MLaaS) paradigm where the data owners (clients) upload their data to the model provider (server), the server may not conduct the unlearning operations faithfully [10, 26]. In this paper, we consider a particular type of dishonest behavior, namely *incomplete edge unlearning*, by which the server removes only a subset of the requested edges from the model. There is an urgent need for developing GU verification mechanisms to ascertain that the server has complied with RTBF and removed all the requested edges from the GNN.

There is a limited number of studies on verification of machine unlearning [12, 26, 30, 41, 47]. While most of these studies focus on machine unlearning models trained on image or text data, none of these works have considered GU models. Intuitively, the link membership inference attacks (LMIAs) against GNNs [14, 36], which have been widely used for evaluation of graph unlearning, can be used for verification. However, the deployment of LMIA requires the client to be equipped with a similar computational power as the model provider [26]. This is not feasible for the MLaaS paradigm. Moreover, due to the inherent node dependencies in GNNs, LMIA struggles to accurately detect whether edges have been completely removed. Our empirical studies will show that LMIA fails to detect incomplete unlearning effectively, demonstrating its limitations as a verification approach.

In this paper, we consider using graph adversarial perturbations for verification. Specifically, the verifier identifies a set of nodes (called *token nodes*) in the graph. Then he injects some fake edges (called *challenge edges*) into the graph. These challenge edges will

change the predictions of token nodes in the "poisoned" model (i.e., the model trained on the data injected with challenge edges). Next, the verifier requests the server to remove the challenge edges from the poisoned model. Then he determines if the server has performed incomplete edge unlearning by observing if the token nodes' predictions output by the unlearning model will be changed.

However, using adversarial edge perturbations for verification of completeness unlearning presents a unique challenge. In addition to requiring the *soundness of perturbations* — meaning that injecting them into the graph must alter the predictions of token nodes — *these perturbations must also be necessary*, meaning that removing only a subset of the challenge edges should not be able to trigger any change on the token nodes' predictions. Ensuring this necessity condition is crucial to effectively detect incomplete unlearning behaviors. Unfortunately, as shown by our empirical study, existing graph adversarial perturbation methods (e.g., [4, 51**?**]) fail to meet this necessity requirement.

Our contributions. In this paper, we introduce PANDA, the first PrabAbilistic iNtegrity eDge unleArning framework, for GU models. We made the following contributions.

Quantifying verification probabilities. We quantify the verification power of PANDA in terms of two probabilities, namely the *true positive probability* (P_{TP}) which is the probability of correctly detecting incomplete unlearning, and the *true negative probability* P_{TN} which is the probability that a fully compliant unlearning model will not be incorrectly determined as incomplete. Based on two probabilities, we formally define our verification goal as providing the (α, β)*-completeness verification guarantee*, which requires $P_{TP} \geq \alpha$, $P_{TN} \geq \beta$, where α, β are user-specified thresholds. Additionally, we derive the number of token nodes required to achieve the (α, β)*-completeness* requirement for any given values of α, β.

Algorithm design. We first define *1-perturbation fragile (1PF) nodes*, which are the nodes that are so sensitive to perturbations that their predictions change when a single adversarial edge is added. We take the 1PF nodes as token nodes, and construct their perturbations as challenge edges. We propose an efficient heuristic algorithm to find the token nodes and construct the challenge edges. One challenge of the edge construction process is that the challenge edges may interfere with one another when added to the graph simultaneously, potentially preventing the change in token nodes' prediction. We tackle this challenge by considering two scenarios: one where the challenge edges are linked to the same token node, and another where they are linked to different token nodes. In both scenarios, we ensure that these challenge edges effectively trigger changes in the predictions of the token nodes, while removing a subset of them does not affect those predictions.

Theoretical analysis. We prove that PANDA always provides (100%, 100%)-completeness guarantee for 1-layer Graph Convolutional Networks (GCNs) with binary node classification task.

Empirical evaluation. We conducted extensive experiments over four state-of-the-art GNN models, three graph datasets, and five graph unlearning algorithms to evaluate PANDA's performance. The results show that PANDA can provide exceptionally high verification probabilities with negligible model accuracy loss, even for complex GNNs and multi-class node classification tasks. Furthermore, PANDA outperforms both the SOTA link membership inference attack [14] and the baseline approaches that leverage the adversarial perturbations generated by SOTA graph adversarial attacks. Additionally, PANDA is robust against the detection of graph adversarial perturbations.

2 RELATED WORK

Graph unlearning. Graph unlearning involves eliminating the impact of specific graph elements such as nodes, edges, labels, and node features from a trained GNN model. In this paper, we mainly focus on *edge unlearning*, i.e., removing the influence of specific edges from a trained GNN model. Recently, some approximate edge unlearning methods are proposed. Broadly speaking, these methods can be categorized into the following types: (1) *Partition-based approaches* partition the training graph into multiple shards and only retrain the shards that contain the edges to be removed [6, 33, 48]; (2) *Influence-based approaches* estimate the influence of the to-be-removed edges on the GNN model and remove the estimated influence by directly updating the model parameters [8, 9, 38, 39]; (3) *Optimization-based approaches* that modify the objective function of the target model with specific types of loss functions to mitigate the impact of unlearning entities [7, 18, 50].

Graph adversarial attacks. The goal of graph adversarial attacks is to generate adversarial perturbations to misguide the victim model. Based on the adversary's goal, the attacks thus can be classified into two categories: (1) *Untargeted attacks* that aim to degrade the overall performance of the classifier over all nodes [42, 52]; and (2) *Targeted attacks* that intend to change the prediction of specific nodes [3, 28, 32, 37, 51]. We refer the readers to some excellent surveys [5, 15, 27] on graph adversarial attacks and defenses for further reading.

Unlearning verification. Broadly, the existing works on verification of machine unlearning can be categorized into two types: *perturbation-based methods* and *proof-based verification*. In particular, the perturbation-based methods add the perturbations (e.g., backdoor triggers [26**?**] and influential sample pairs [41]) to the training data and leverage model predictions by the model trained on poisoned data versus clean data to detect whether the server has faithfully removed the perturbations. In contrast, the proof-based approaches leverage cryptographic techniques to provide a proof of unlearning (PoUL). The proof records the operations for unlearning so that data owners can reproduce every unlearning step from the proof [30]. The PoUL can be constructed in the trusted execution environments [34]. All these existing works consider ML models trained on non-graph data. We are the first to consider verification of GU models. In particular, we will design perturbation-based verification approaches for GU models.

3 PROBLEM DEFINITION

In this paper, we consider a Machine-Learning-as-a-Service (MLaaS) scenario [21, 46] that consists of two parties: (i) a *client* (denoted as C) who owns a training graph $\mathcal{G}$, and (ii) an MLaaS *service provider* (denoted as $\mathcal{S}$) who trains a GNN model on $\mathcal{G}$. For any given node $v \in \mathcal{G}$, the GNN model outputs a *posterior probability vector* for v, in which the i-th entry indicates the probability that v is associated with the i-th label. In this paper, we consider GNNs trained in the transductive setting.

Threat model. We consider C as the victim and $\mathcal{S}$ as the adversary. $\mathcal{S}$ is honest on training the model on the dataset provided

by the client. However, it will attempt to cheat on unlearning for various reasons, such as avoiding the costly computations required [2, 11] or being unwilling to tolerate the potential degradation in model performance following data removal [10, 24]. Therefore, $\mathcal{S}$ ignores some edge removal requests by C, leading to *incomplete edge unlearning*. Formally, given a set of edges E_U requested to be removed from the model, $\mathcal{S}$ removes only a subset $E'_U \subset E_U$ from the model, and returns the resulting model to C. We assume the edges in E_U have an equal probability of being selected to be retained in the model.

Verification goal. Given a graph $\mathcal{G}$, a GNN model f trained on $\mathcal{G}$, a set of edges $E_U \subset \mathcal{G}$ to be removed from f, and an edge unlearning algorithm $\mathcal{A}_{\text{UL}}$, let f_{U} be the GU model returned by the server. The verifier aims to decide whether f_{U} is achieved by incomplete unlearning, i.e., $\exists E'_U \subset E_U,\ f_{\text{U}} = \mathcal{A}_{\text{UL}}(f, E'_U)$. As a starting point for investigating GU verification, we consider model retraining as the unlearning algorithm. The effectiveness of PANDA in detecting approximate edge unlearning models will be evaluated through empirical studies (Section 6).

Verifier's capabilities. We assume the existence of a "verifier" (denoted as $\mathcal{V}$), who can be either C himself or a third party trusted by C. $\mathcal{V}$ has two types of capabilities: (i) **Graph perturbation** - $\mathcal{V}$ has full access to the training graph $\mathcal{G}$ and can insert adversarial perturbations in the format of fake edges for verification purpose. However, there exists a *perturbation budget* which limits the amount of fake edges. Specifically, $\mathcal{V}$ can insert up to B poisoning edges into the graph, where B is a user-specified positive integer. (ii) **Black-box model access** - $\mathcal{V}$ has no knowledge about the architecture of the target model nor the unlearning algorithm. However, $\mathcal{V}$ is able to query both f and f_{U}, and in response, receives the corresponding posterior probability vectors.

4 VERIFICATION FRAMEWORK

Our goal is to leverage adversarial perturbations to verify the completeness of GU models. Specifically, the verification mechanism consists of two phases (Figure 1):

- **Preparation phase**: given a graph $\mathcal{G}(V, E)$ with a set of nodes V and a set of edges E, the verifier $\mathcal{V}$ picks a set of nodes $V^\star \subset V$ as *token nodes*. For each token node $v \in V^\star$, $\mathcal{V}$ generates a set of fake edges $E^\star_v \not\subset E$ whose insertion into $\mathcal{G}$ alters v's prediction. We call $E^\star = \bigcup_{\forall v \in V^\star} E^\star_v$ the *challenge edges*. $\mathcal{V}$ injects $E^\star$ into $\mathcal{G}$ and sends $\widetilde{\mathcal{G}}(V, E \cup E^\star)$ to $\mathcal{S}$. $\mathcal{S}$ trains a GCN on $\widetilde{\mathcal{G}}$ and obtains the *challenge model*, denoted as f_{C}.
- **Verification phase**: $\mathcal{V}$ requests $\mathcal{S}$ to remove all edges in $E^\star$ from f_{C}. Let f_{U} be the unlearning model returned by $\mathcal{S}$. $\mathcal{V}$ determines that f_{U} passes verification if f_{C} and f_{U} output different labels for each token node. Otherwise, $\mathcal{V}$ concludes the server cheats on the GU process.

To catch incomplete edge unlearning, the challenge edges should satisfy two requirements: (i) **Soundness**: Injecting challenge edges into the training graph alters the predictions of each token node output by f_{C}; and (ii) **Necessity**: Removing any subset of the challenge edges from f_{C} cannot change the predictions of any token node. However, the randomness of GNNs makes it impossible to construct challenge edges that are always sound and/or necessary. Therefore, we define two probabilities, namely the *soundness probability* ($\mathbf{p_S}$) and *necessity probability* ($\mathbf{p_N}$), to quantify the likelihood that they are sound and necessary for verification. Specifically, p_S measures the likelihood that the token nodes' prediction will be changed by injection of challenge edges, and p_N measures the likelihood that the predictions of token nodes remain unchanged by removing only a subset of challenge edges from the model. The details of how to measure p_S and p_N empirically will be provided in Section 5.4.

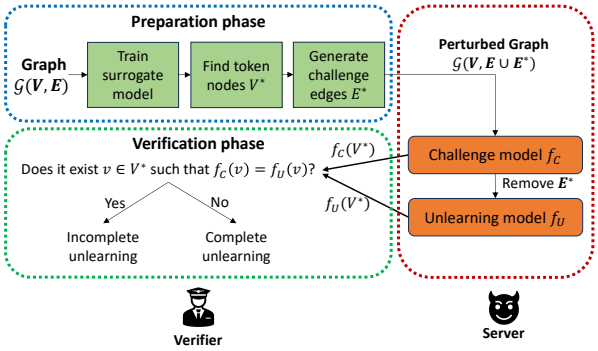


Figure 1: Overview of PANDA. $f_{\text{C}}(v)$ and $f_{\text{U}}(v)$ denote the classification label of the node v predicted by f_{C} and f_{U}.

After we obtain both p_S and p_N, we compute two probabilities to quantify the confidence of using adversarial perturbations for verification: (i) **True positive probability $\mathbf{P_{TP}}$**: the probability of catching incomplete edge unlearning; (ii) **True negative probability $\mathbf{P_{TN}}$**: the probability that the verification mechanism correctly flags a complete edge unlearning model. Next, we explain how to measure these two probabilities.

Following our verification strategy, an incomplete unlearning model can evade verification if the prediction of all token nodes change when only a subset of $E^\star$ is removed. Given that the probability of a single token node's prediction changing due to incomplete unlearning is $(1 - p_N)$, where p_N is the necessity probability of the challenge edges, the probability that an incomplete unlearning model can pass the verification with m token nodes is $(1 - p_N)^m$. Therefore, P_{TP} is measured as follows:

$$P_{TP} = 1 - (1 - p_N)^m. \tag{1}$$

On the other hand, a model obtained by complete unlearning will be correctly verified as complete if the prediction of all token nodes changes in the unlearning model. Given that the probability of a token node's prediction being altered by the challenge edges is p_S, the *true negative probability* P_{TN} is measured as follows:

$$P_{TN} = (p_S)^m. \tag{2}$$

Based on the soundness and necessity probabilities, we formally define our verification goal, which is quantified as (α, β)-*completeness*:

Definition 4.1 ((α, β)-**completeness**). *Given a graph $\mathcal{G}$, a GNN f trained on $\mathcal{G}$, a set of edges $E_U \subset \mathcal{G}$ requested to be removed from f, and the model f_{U} to be verified for unlearning completeness, we say a probabilistic verification mechanism provides* (α, β)-completeness *guarantee of f_{U} if $P_{TP} \geq \alpha$ and $P_{TN} \geq \beta$, where $\alpha, \beta \in [0, 1]$ are user-specified thresholds.*

Based on Equations (1) and (2), we derive the range for the number of token nodes required to achieve (α, β)-completeness for

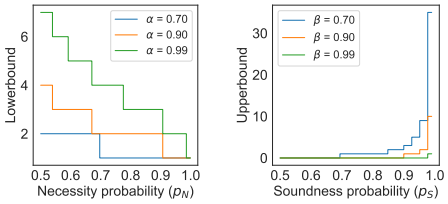


Figure 2: Lowerbound & upperbound of the number of token nodes to satisfy (α, β)-completeness guarantee.

given values of α and β as follows:

$$\lceil \log_{1-p_N}(1-\alpha) \rceil \leq m \leq \lfloor \log_{p_S} \beta \rfloor. \tag{3}$$

Figure 2 illustrates how p_S, p_N, α, and β impact the lower/upper bound of m. Notably, m may not exist when $\lceil \log_{1-p_N}(1-\alpha) \rceil > \lfloor \log_{p_S} \beta \rfloor$. This situation can occur when either p_S or p_N is low, even when α and β are not high. For instance, when $p_S = 0.7$, $p_N = 0.3$, $\alpha = 0.7, \beta = 0.7$, then $\log_{1-p_N}(1-\alpha) = 3.38$ and $\log_{p_S} \beta = 1$, resulting in no values that satisfy Eqn. (3). Our empirical studies will show that the adversarial edges generated by the state-of-the-art graph adversarial attacks [4, 51**?**] always have a very low p_N value (no higher than 50%), leading to their failure to satisfy the (α, β)-completeness requirement. Therefore, we aim to construct the challenge edges that have both high p_S and p_N values.

When $\lceil \log_{1-p_N}(1-\alpha) \rceil \leq \lfloor \log_{p_S} \beta \rfloor$, we select the smallest m, i.e., $m = \lceil \log_{1-p_N}(1-\alpha) \rceil$, as the number of required token nodes. This is because increasing the number of token nodes results in a greater number of challenge edges being injected, which can lead to a higher accuracy loss for the target model. Note that even with a high α, the required number of token nodes remains relatively small. For example, with $\alpha = 0.99$, no more than 7 token nodes are needed even when $p_N = 0.5$.

So far, we assume that the edge unlearning requests consist solely of challenge edges. We extend this analysis to the scenario when the unlearning request includes both real and challenge edges, and derive its probabilistic verification probabilities. The details can be found in Section 7.1.

5 METHODOLOGY

In this section, we present the details of PANDA, our verification mechanism. PANDA consists of three steps: (1) training a surrogate model; (2) identifying the token nodes $V^\star$; and (3) generating the challenge edges $E^\star$ of $V^\star$. First, we discuss the details of each step. Then we explain how to make the 3-step process provide (α, β)-completeness guarantee.

5.1 Step 1: Training Surrogate Model

As the verifier $\mathcal{V}$ only has black-box access to the target GNN, it cannot find the token nodes in the training graph easily. Therefore, $\mathcal{V}$ employs a *surrogate model*, which mimics the behaviors of the target model, to find token nodes. To train the surrogate model (denoted as f_S), $\mathcal{V}$ generates a surrogate training graph $\mathcal{G}_S$ that has the same graph structure as the target graph $\mathcal{G}$. This is feasible as $\mathcal{V}$ has full access to the training graph. However, to ensure f_S mimics the behavior of the target model, the nodes in $\mathcal{G}_S$ are associated with the labels predicted by the target model f obtained through the black-box access to f.

5.2 Step 2: Finding Token Nodes

Intuitively, challenge edges are likely to be both sound and necessary if they are linked to nodes that are highly sensitive to perturbations. Following this intuition, we define *1-perturbation fragile (1PF) nodes* as token nodes that are so sensitive to perturbations that the addition of a single adversarial edge will change their predictions. Formally, given a graph $\mathcal{G}(V, E)$, we say a node $v \in V$ is *1-perturbation fragile* (1PF) if there exists a fake edge $e' \notin \mathcal{G}$ such that connecting it with v makes $y_v \neq y'_v$, where y_v and y'_v are v's prediction by the model trained on $\mathcal{G}(V, E)$ and $\mathcal{G}(V, E \cup \{e'\})$, respectively. The edge e' is referred to as a *1-edge perturbation* of v.

Based on the definition of 1PF nodes, we formalize the problem of finding 1PF nodes as follows:

Problem 1. [Finding **1PF** nodes] *Given a graph $\mathcal{G}$ and its adjacency matrix $\mathbf{A}$, let $\widetilde{\mathcal{G}}$ be the graph after adding perturbation to $\mathcal{G}$ and $\tilde{\mathbf{A}}$ be the adjacency matrix of $\widetilde{\mathcal{G}}$. Let y and $\tilde{y}$ be v's label predicted by the GNN trained on $\mathcal{G}$ and $\widetilde{\mathcal{G}}$, respectively. Then the worst-case margin of v between y and $\tilde{y}$, denoted as $m^v(y, \tilde{y})$, is:*

$$\begin{aligned} m^v(y, \tilde{y}) :&= \underset{\tilde{\mathbf{A}}}{maximize}\ f_\theta(v, \tilde{y}) - f_\theta(v, y) \\ subject\ to\ \theta &= \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\theta; \tilde{\mathbf{A}}, \mathbf{X}),\ ||\tilde{\mathbf{A}} - \mathbf{A}||_0 = 2, \\ \tilde{\mathbf{A}}_{ij} &\geq \mathbf{A}_{ij}\ (\forall 1 \leq i, j \leq n) \end{aligned} \tag{4}$$

If $m^v(y, \tilde{y}) > 0$ for at least one label $\tilde{y} \neq y$, v is a 1PF node w.r.t. $\tilde{\mathbf{A}}$.

Specifically, $||\tilde{\mathbf{A}} - \mathbf{A}||_0 = 2$ guarantees that only one perturbation has occurred, and $\tilde{\mathbf{A}}_{ij} \geq \mathbf{A}_{ij}$ ensures that the perturbation involves only the addition of edges. As the verifier uses a surrogate model f_S, f is substituted by f_S. Note that our 1PF problem fundamentally differs from the existing graph adversarial perturbation method [54]: while [54] requires $m^v(y, \tilde{y}) > 0$ for all $\tilde{y} \neq y$, 1PF requires $m^v(y, \tilde{y}) < 0$ for at least one $\tilde{y} \neq y$. Thus our 1PF problem cannot be easily solved by [54].

Solving the 1PF problem (Eqn. (4)) has two challenges: (1) The binary nature of the adjacency matrix makes the problem intractable; (2) The optimization problem is non-convex due to the nonlinear activation function in GNNs. We follow two strategies to address these challenges: (1) transforming the binary adjacency matrix to a continuous-valued one; (2) constructing relaxing the activation function. Next, we discuss the details of the strategies.

(1) Transforming adjacency matrix. We transform the binary adjacency matrix $\mathbf{A}$ into a continuous-valued message-passing matrix (denoted as $\mathring{\mathbf{A}}$), with each entry transformed as follows [54]:

$$\mathring{\mathbf{A}}_{ij} = \begin{cases} \frac{1}{\sqrt{d_i d_j}}, & \text{if } \mathbf{A}_{ij} = 1 \vee i = j \\ 0, & \text{Otherwise} \end{cases} \tag{5}$$

where d_i, d_j denote the degree of nodes v_i and v_j, respectively. In the following discussion, we use $\mathring{\mathbf{A}}$ and $\mathring{\tilde{\mathbf{A}}}$ to denote the continuous-valued message-passing matrices of the original matrix $\mathbf{A}$ and the perturbed matrix $\tilde{\mathbf{A}}$, respectively.

Replacing $\tilde{\mathbf{A}}$ with $\mathring{\tilde{\mathbf{A}}}$ in the optimization problem (Eqn. (4)) raises two issues: (1) how to transform the constraint $||\tilde{\mathbf{A}} - \mathbf{A}||_0 = 2$

according to $\mathring{\tilde{\mathbf{A}}}$ (i.e., deriving $||\mathring{\tilde{\mathbf{A}}} - \mathring{\mathbf{A}}||_0$)? (2) how to rewrite the edge-addition-only constraint $\tilde{\mathbf{A}}_{ij} \geq \mathbf{A}_{ij}$ according to $\mathring{\tilde{\mathbf{A}}}$? Next, we discuss our solution to these two issues.

Transforming $||\tilde{\mathbf{A}} - \mathbf{A}||_0 = 2$ constraint. Enlightened by [54], we derive three types of bounds:

- *Lower and upper element-wise bounds* (denoted as L_{ij} and U_{ij}) for each element $\mathring{\tilde{\mathbf{A}}}[i, j]$ in $\mathring{\tilde{\mathbf{A}}}$;
- *Lower and upper row-wise bounds* (its lower and upper bounds (denoted as L_i^{row} and U_i^{row}) for each row $\mathring{\tilde{\mathbf{A}}}[i]$ in $\mathring{\tilde{\mathbf{A}}}$;
- The *upper global bound* of $||\mathring{\tilde{\mathbf{A}}} - \mathring{\mathbf{A}}||_0$ (denoted as U^{glob}). We do not consider the lower global bound as its inclusion would make the optimization problem non-convex.

Next, we present the results of these bounds.

$$
\begin{aligned}
L_{ij} &= \min(\mathring{\mathbf{A}}_{ij}, \frac{1}{\sqrt{d_i+1}\sqrt{d_j+1}}),\ U_{ij} = \max(\mathring{\mathbf{A}}_{ij}, \frac{1}{\sqrt{d_i+1}\sqrt{d_j+1}}),\\
L_i^{\text{row}} &= \min(\frac{1}{\sqrt{d_i+1}} \sum_{\substack{j\neq i\\ j\in\mathcal{N}_i}} \frac{1}{\sqrt{d_j}} + \frac{1}{\sqrt{d_i+1}\sqrt{d_{max}+1}} + \frac{1}{d_i+1},\\
&\quad \frac{1}{\sqrt{d_i}} \sum_{j\in\mathcal{N}_i\setminus\{a,b\}} \frac{1}{\sqrt{d_j}} + \frac{2}{\sqrt{d_i}\sqrt{d_{max}+1}} + \frac{1}{d_i}),\\
U_i^{\text{row}} &= \max(\frac{1}{\sqrt{d_i+1}} \sum_{\substack{j\neq i\\ j\in\mathcal{N}_i}} \frac{1}{\sqrt{d_j}} + \frac{1}{\sqrt{d_i+1}\sqrt{d_{min}+1}} + \frac{1}{d_i+1},\\
&\quad \frac{1}{\sqrt{d_i}} \sum_{\substack{j\neq i\\ j\in\mathcal{N}_i}} \frac{1}{\sqrt{d_j}} + \frac{1}{d_i}),\\
U^{\text{glob}} &= 4(\frac{1}{d_{min}} - \frac{1}{d_{min}+1}) + 2 \sum_{\substack{j\notin\{a,b\}\\ j\in\mathcal{N}_{min}}} \frac{1}{\sqrt{d_j}}(\frac{1}{\sqrt{d_{min}}} - \frac{1}{\sqrt{d_{min}+1}}),
\end{aligned}
\tag{6}
$$

where a, b are the two nodes of the perturbed edge, d_{max} and d_{min} are the maximum and minimum node degree in $\mathcal{G}$, and $\mathcal{N}_{min}$ is the L-hop neighborhood of the node that has the lowest node degree, where L is the number of layers of the model. The details of how these bounds are derived can be found in Appendix A.

Transforming $\tilde{\mathbf{A}}_{ij} \geq \mathbf{A}_{ij}$ constraint. To replace this constraint, we derive the lower and upper bounds of the number of "positive" entries in $\mathring{\tilde{\mathbf{A}}} - \mathring{\mathbf{A}}$, denoted as L_{pos} and U_{pos}:

$$
L_{\text{pos}} = \frac{2}{d_{max}+1}, \quad U_{\text{pos}} = \frac{2}{d_{min}+1}. \tag{7}
$$

The derivation of both bounds can be found in Appendix A.

Putting all constraints together. We refine the search space of the perturbed adjacency matrix for the optimization problem (Eqn. (4)) by the transformed constraints as follows:

$$
\begin{aligned}
\Phi(\mathbf{A}) = \Big\{ &\mathring{\tilde{\mathbf{A}}} \in [0,1]^{n\times n} \mid \mathring{\tilde{\mathbf{A}}} = \mathring{\tilde{\mathbf{A}}}^{\intercal}, |\mathring{\tilde{\mathbf{A}}} - \mathring{\mathbf{A}}| \leq U^{\text{glob}},\\
&\forall i,j: L_{ij} \leq \mathring{\tilde{\mathbf{A}}}_{ij} \leq U_{ij},\ \ \forall i: L_i^{\text{row}} \leq \mathring{\tilde{\mathbf{A}}}_i \leq U_i^{\text{row}},\\
&\forall i,j: \mathring{\tilde{\mathbf{A}}}_{ij} > \mathring{\mathbf{A}}_{ij}, L_{\text{pos}} \leq \sum_{i,j} (\mathring{\tilde{\mathbf{A}}}_{ij} - \mathring{\mathbf{A}}_{ij}) \leq U_{\text{pos}}. \Big\}
\end{aligned}
\tag{8}
$$

where $\mathring{\tilde{\mathbf{A}}} = \mathring{\tilde{\mathbf{A}}}^{\intercal}$ ensures that $\mathring{\tilde{\mathbf{A}}}$ is symmetric.

(2) Convex relaxation. As the nonlinearity of the ReLU activation function of the model still hinders solving the optimization problem, we follow [35, 53] to obtain a linear relaxation of the ReLU activation. After matrix transformation and convex relaxation, the optimization problem (Eqn. (4)) is expressed as a jointly constrained bilinear program [1]. We employ the branch-and-bound algorithm [54] to solve the bilinear program[1]. The output of the algorithm includes both the 1PF nodes and their 1-edge perturbations.

5.3 Step 3: Generating Challenge Edges

Verifying the completeness of edge unlearning requires multiple challenge edges. However, due to node dependency in graphs, these 1-edge perturbations can interfere with each other when they are inserted into the graph altogether and thus fail to change some token nodes' predictions. To address this challenge, we consider two types of challenge edges: those associated with the same token nodes and those across different token nodes.

Challenge edges associated with the same token node. First, we define the concept of *consistent 1-edge perturbations*.

Definition 5.1 (Consistent 1-edge perturbations). Given a graph $\mathcal{G}(V, E)$, a 1PF node $v \in \mathcal{G}$, and a set of edges $E_{\mathcal{A}}$, where each edge $e_i \in E_{\mathcal{A}}$ is a 1-edge perturbation of v, we say $E_{\mathcal{A}}$ is a set of *consistent 1-edge perturbations* of v if inserting each 1-edge perturbation $e_i \in E_{\mathcal{A}}$ individually into $\mathcal{G}$ changes v's prediction to be the same label.

We have the following theorem to show that, for 1-layer GCNs with binary node classification tasks, a set of consistent 1-edge perturbations associated with the same token node is guaranteed to be sound and necessary, i.e., inserting all such edges altogether still be able to change the token node's prediction, while deleting only a subset of these edges from the perturbed model cannot change the token node's prediction.

Theorem 5.2. *Given a graph $\mathcal{G}$ and a token node $v \in \mathcal{G}$, let E_P be a set of consistent 1-edge perturbations of v. Then for any 1-layer GCN with a binary node classification task, it is guaranteed that $P_S = P_N = 100\%$, where P_S and P_N are the soundness and necessity probabilities of E_P, respectively.*

The proof of Theorem 5.2 can be found in Appendix B.1.

Challenge edges across different token nodes. To ensure that inserting multiple 1-edge perturbations across different token nodes is still able to change the prediction of each token node, we pick the token nodes that are located outside the L-hop neighborhood of each other, where L is the number of layers of the GNN model. We call these token nodes *independent token nodes*, as each node can only influence its neighbors within its L-hop neighborhood in a GNN model of L layers. We have the following theorem:

Theorem 5.3. *Given a graph $\mathcal{G}$ and a set of independent token nodes $V^\star \subset \mathcal{G}$, let $\mathcal{E} = \cup_{v\in V^\star} E_v$, where E_v is a set of consistent 1-edge perturbations of v. Then for any 1-layer GCNs with binary node classification tasks, using $\mathcal{E}$ as the challenge edges guarantees to provide (100%, 100%)-completeness if the token nodes in $V^\star$ remain to be independent to each other after inserting $\mathcal{E}$ into $\mathcal{G}$.*

The proof of Theorem 5.3 can be found in Appendix B.2. While both Theorems 5.2 and 5.3 only apply to 1-layer GCNs with binary classification tasks, our experiments will demonstrate that PANDA is able to provide high verification probabilities for GNNs with

[1]We use CPLEX solver (https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer) to implement the branch-and-bound algorithm.

more complex architecture as well as multi-class classification tasks (Section 6.3).

5.4 Achieving Verification Guarantee

The key to satisfying the (α, β)-completeness guarantee is to calculate the number of required token nodes m (Eqn. (3)). As the calculation of m requires both p_S and p_N values, we first estimate both p_S and p_N empirically. Specifically, we identify a set of token nodes (50 nodes in our experiments) and construct a set of challenge edges for these token nodes (250 challenge edges, with 5 edges per token node in our experiments). Then we measure p_S as the percentage of the selected token nodes whose prediction labels are changed by injecting challenge edges into the graph, and p_N as the percentage of the token nodes whose prediction remains unchanged unless all challenge edges are removed from the trained model. To provide statistical validity, we repeat the process multiple times (5 times in our experiments), and calculate p_S, p_N as the average across all the trials. After obtaining both p_S and p_N values, we compute m, i.e., the number of token nodes required to provide the (α, β)-completeness guarantee, by following Eqn. (3). After we obtain the value of m, we follow the 3-step process to identify the token nodes and generate the challenge edges. The pseudo code can be found in Appendix C.

A corner case of PANDA is that it may not be able to find the required m 1PF nodes. A potential solution is to relax the requirement of the token nodes to be *kPF nodes*, i.e., those nodes whose predictions change by the addition of $k \geq 1$ edges. We leave the work of verification using kPF node perturbations to future work.

6 EXPERIMENTS

This section presents the results of our empirical evaluation, aiming to address three research questions:

- **RQ$_1$**: How effective and efficient is PANDA for completeness verification?
- **RQ$_2$**: How do various factors impact the performance of PANDA?
- **RQ$_3$**: How does PANDA perform across various GU models? Additionally, how does PANDA perform when the surrogate model and target model have different architectures?
- **RQ$_4$**: Is PANDA robust against detection of adversarial edges?

6.1 Setup

All the experiments are performed on a server with eight NVIDIA A100 GPUs (40GB memory). Each experiment is repeated five times, and the average results are reported.

Datasets. We utilize three benchmark datasets, namely the *CiteSeer* [45], *LastFM* [22], and *CS* [25] datasets. The statistics of these datasets can be found in Appendix D.

GNN models. We consider four mainstream GNNs: GCN [16], GAT [31], GraphSAGE [13], and GIN [43]. All these GNN models are set up as a 2-layer architecture, with 16 neurons at each layer (i.e., $2L \times 16N$). We follow the 5:1:4 ratio to split the input datasets into the training, validation, and testing sets, respectively.

Evaluation metrics. To assess the verification effectiveness, we measure both *True Positive Probability* $\mathbf{P_{TP}}$ and *True Negative Probability* $\mathbf{P_{TN}}$ of our verification mechanism. Specifically, we conduct 50 *positive trials* simulating incomplete unlearning by the dishonest server, and 50 *negative trials* representing complete unlearning by the honest server. For each incomplete unlearning trial, we randomly select $s\%$ (called the **incompleteness ratio**) of challenge edges and do not perform unlearning on these selected edges. We run PANDA on these trials and evaluate both P_{TP} and P_{TN}, where P_{TP} and P_{TN} are measured as the percentage of positive and negative trials that are flagged as positive and negative by verification, respectively. Intuitively, higher P_{TP} and P_{TN} values indicate higher verification effectiveness.

As PANDA perturbs the training graph, we also measure the *accuracy loss* of the GNN model incurred by injection of challenge edges: $AccLoss = \frac{\text{Acc}-\text{Acc}'}{\text{Acc}}$, where Acc and Acc′ are the node classification accuracy before and after inserting challenge edges.

Methods for comparison. As we are the first work on GU verification, there is no established method for comparison. Therefore, we employ three SOTA graph poisoning attacks, namely *Nettack* [51], *SGAttack* [**?**], and *FGA* [4], to generate adversarial edges as the challenge edges. We evaluate the effectiveness of utilizing these adversarial edges for verification in comparison with PANDA.

Besides perturbation-based verification, we consider *link membership inference attacks (LMIAs)* as an additional baseline for comparison. In particular, we leverage StealLink [14], a SOTA LMIA, to infer the membership of the removed edges in the unlearning model. We flag a trial as positive (i.e., incomplete unlearning) if at least one of the removed edges is inferred as a member by LMIA, and as negative otherwise. Then we measure P_{TP} and P_{TN} of LMIA in the same way as PANDA.

6.2 Verification Performance (RQ$_1$)

In this section, we present the verification performance results of PANDA using GCN as the target model and retraining as the unlearning method. Results for other GNN types and unlearning algorithms are provided in the transferability study (Section 6.4).

Verification effectiveness. Table 1 presents the effectiveness of PANDA in terms of both P_{TP} and P_{TN}. Overall, PANDA consistently achieves high performance across all the datasets, with P_{TP} ranging from 97.6% to 99.2% and P_{TN} ranging from 92.8% to 100%. We attribute the high verification accuracy of PANDA to the soundness and necessity of the token nodes, supported by their high soundness and necessity probabilities (p_S, p_N), which reach up to 100% and 78.8%, respectively. Further details on the estimated values of p_S and p_N of token nodes can be found in Appendix E.1.

Comparison with baselines. First, we compare PANDA with the three baseline methods (Nettack, SGAttack, FGA) that leverage graph poisoning in terms of their P_{TP} and P_{TN}, and report the results in Table 1 [2]. The results show that PANDA significantly outperforms these three baseline methods, with the improvement by up to 45% on P_{TP} and 450% in P_{TN}. Indeed, with the same number of token nodes, only PANDA consistently meets the $(90\%, 90\%)$-completeness requirement across all settings. These results demonstrate that the challenge edges generated by the three SOTA graph poisoning attacks are not suitable for verification.

Next, we compare P_{TP} and P_{TN} of the LMIA baseline with PANDA, and include the results in Appendix E.2 due to the limited space. We observe that the LMIA baseline fails to be an effective

[2]Due to the low p_N values of the three baseline methods, none of them can find a suitable m value for $\alpha, \beta = 90\%$. Thus, we use the same number of token nodes for PANDA and the baseline methods.

Table 1: Verification performance of PANDA and baselines. The best results are marked with green.

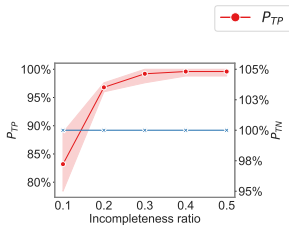
Method	CiteSeer			LastFM			CS		
	P_{TP}	P_{TN}	AccLoss	P_{TP}	P_{TN}	AccLoss	P_{TP}	P_{TN}	AccLoss
Nettack	67.2% ± 0.059	85.2% ± 0.064	0.157%	93.6% ± 0.038	53.2% ± 0.077	0.067%	87.2% ± 0.041	58.8% ± 0.056	0.028%
SGAttack	86.0% ± 0.032	45.2% ± 0.066	0.168%	96.4% ± 0.017	16.8% ± 0.046	0.068%	87.2% ± 0.046	34.8% ± 0.061	0.029%
FGA	70.8% ± 0.039	73.6% ± 0.065	0.151%	90.0% ± 0.037	48.0% ± 0.120	0.065%	82.8% ± 0.086	38.8% ± 0.087	0.027%
PANDA (ours)	97.6% ± 0.017	99.6% ± 0.009	0.134%	97.6% ± 0.015	92.8% ± 0.037	0.063%	99.2% ± 0.011	100% ± 0	0.026%

Dataset	Required guarantee α			
	70%	80%	90%	95%
CiteSeer	85.0%	100%	98.5%	100%
LastFM	78.8%	92.0%	97.2%	98.8%
CS	93.0%	99.0%	100%	100%

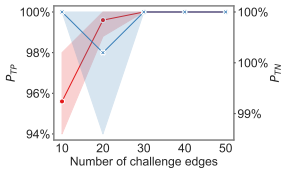
(a) Varying α value ($\beta = 90\%$)

Dataset	Required guarantee β			
	70%	80%	90%	95%
CiteSeer	100%	100%	100%	100%
LastFM	93.2%	92.0%	92.8%	93.6%
CS	100%	100%	100%	100%

(b) Varying β value ($\alpha = 90\%$)



(c) Incompleteness ratio



(d) Perturbation budget

Figure 3: Impact of α, β, incompleteness ratio, and perturbation budget on the performance of PANDA.

Table 2: Verification time (seconds), LastFM dataset, $\beta = 90\%$.

Guarantee α	PANDA			Local verification
	Train surro. model	Generate perturbation	Verification	
70%	2.88	2.48	0.41	18.93
80%		2.03	0.61	26.80
90%		1.77	0.81	31.75
95%		1.75	1.00	37.87

verification method, as it always flags all trials positive (i.e., it always infers at least one removed edge as a member), and thus consistently offers 100% P_{TP} and 0% P_{TN}. The failure of the LMIA baseline comes from the fact that it is difficult to completely remove graph components from the GNNs due to the node dependency in the graph [6]. This finding validates the necessity of our poisoning-based verification mechanism.

Accuracy loss by verification. Table 1 ("AccLoss" column) reports the model accuracy loss incurred by injection of challenge edges. We observe that PANDA exhibits negligible accuracy loss, up to 0.134%, across all the settings. Additionally, although the accuracy loss of PANDA and the baselines are comparable as they insert the same amounts of challenge edges, PANDA is able to marginally outperform the baselines in terms of accuracy loss.

Verification efficiency. To evaluate the efficiency of PANDA, we measure the time taken for each step, namely training the surrogate model, generating perturbations, and verification. We compare these results with the time performance of local verification, i.e., model retraining by the client. The results for the LastFM dataset are presented in Table 2, while results for other datasets can be found in Appendix E.3. Overall, PANDA is highly efficient, with the verification time as low as around one second, even for a high verification guarantee of 95%. Indeed, PANDA significantly outperforms local verification, with the verification time being up to 46 times faster. Even when considering the time required for both surrogate model training and generating perturbations, PANDA still demonstrates a substantial speedup, achieving a 6.7x improvement over local verification. These results show that PANDA can efficiently verify unlearning models while maintaining high verification accuracy.

6.3 Factor Analysis (RQ_2)

In this section, we examine how various factors - including α and β, the incompleteness ratio, the number of challenge edges, and the complexity of the GNN model - affect the performance of PANDA. Besides these factors, we also investigate the impact of the number of token nodes on PANDA's performance, and include the results in Appendix E.4.

α and β. We vary the values of α and β for the (α, β)-completeness requirement, and measure both P_{TP} and P_{TN}. Our goal is two-fold: (1) examine how α and β impact P_{TP} and P_{TN}; (2) validate if PANDA always can provide the (α, β)-completeness guarantee, i.e., whether $P_{TP} \geq \alpha$ and $P_{TN} \geq \beta$ in all the settings. Figure 3 (a) and (b) report the results. Notably, higher α and β values lead to higher P_{TP} and P_{TN} values. Furthermore, PANDA meets the (α, β)-completeness requirement (i.e., $P_{TP} \geq \alpha$ and $P_{TN} \geq \beta$) across all the settings. Specifically, P_{TN} is as high as 100% on the CiteSeer and CS dataset, and at least 92% on the LastFM dataset. This demonstrates the effectiveness of PANDA in verifying incomplete edge unlearning.

Incompleteness ratio. We measure the verification performance of PANDA and the baseline methods under various incompleteness ratios. Figure 3 (c) reports the P_{TP} result of PANDA on the CiteSeer dataset[3]. We observe that P_{TP} improves when the incompleteness ratio increases from 0.1 to 0.5. Remarkably, PANDA consistently exhibits a high P_{TP} value across all the settings, reaching nearly 100% when the incompleteness ratio exceeds 0.2. It still surpasses 80%, even with an incompleteness ratio as small as 0.1.

Perturbation budget. We vary the perturbation budget (i.e., the total number of challenge edges) from 10 to 50. As shown in Figure

[3] P_{TN} stays the same in Figure 3 (c) for various incompleteness ratios as P_{TN} is measured against the complete unlearning trials (i.e., negative trials) only.

Table 3: Impact of GCN complexity and type of classification tasks on the performance of PANDA (CiteSeer dataset). BC & MC denote binary and multi-class classification.

	1 layer		2 layers		3 layers	
	BC	MC	BC	MC	BC	MC
P_{TP}	100%	100%	92.4%	94.4%	91.6%	91.2%
P_{TN}	100%	100%	100%	100%	87.2%	94.8%

Table 4: Performance of PANDA across different GU models

	CEU	Eraser	GDelete	MEGU
P_{TP}	93.2%	88.0%	42.0%	84.0%
P_{TN}	70.8%	46.4%	86.0%	82.4%

3 (d), PANDA consistently maintains a high P_{TP} value across all settings, increasing from 95.4% to 100% as the number of challenge edges rises. Similarly, PANDA achieves a significant P_{TN}, ranging from 98% to 100%.

Complexity of GCNs & type of classification tasks. Our theoretical results (Theorem 5.3) are specific to 1-layer GCNs with binary classification tasks. To extend our evaluation, we test PANDA on GCNs of greater complexity and with multi-class classification tasks. We set up three GCNs: a 1-layer GCN, a 2-layer GCN, and a 3-layer GCN, with 16 neurons per layer. Table 3 displays the performance of PANDA on these GCNs trained on the Citeseer dataset. Additional results for the other two datasets are available in Appendix E.5. We have the following observations. For 1-layer GCNs with binary classification tasks, PANDA consistently achieves 100% P_{TP} and P_{TN}, confirming our theoretical results. For 2-layer GCNs, PANDA maintains a minimum P_{TP} of 92.4% and a P_{TN} of 100%. In the case of 3-layer GCNs, while there is a slight decrease in both P_{TP} and P_{TN}, they remain above 91.6% and 87.2%, respectively. Similar performance is observed for multi-class classification tasks, with both P_{TP} and P_{TN} no lower than 91.2% in all the settings. These results demonstrate that PANDA can effectively detect incomplete edge unlearning with a high probabilistic guarantee, even for more complex GCNs and multi-class node classification tasks.

6.4 Transferability of PANDA (RQ_3)

In this part of the experiments, we evaluate the performance of PANDA across different settings.

Across different GNN models. We measure the performance of PANDA when the surrogate model and the target model have different architectures (e.g., the target model is GAT and the surrogate model is GCN). We observe that PANDA remains to be effective when transferring across different types of GNN models, with P_{TP} and P_{TN} no lower than 92% and 93.2%, respectively. We include the results in Appendix E.6, due to the page limit.

Across different GU methods. We consider four state-of-the-art approximate edge unlearning approaches: *Eraser* [6][4], *CEU* [39][5], *GDelete* [7][6], and *MEGU* [18][7]. Table 4 reports both P_{TP} and P_{TN} of PANDA across these methods. The main observation is that the performance of PANDA varies across the four unlearning models. Specifically, MEGU demonstrates both high P_{TP} and P_{TN} values. CEU achieves a higher P_{TP} but a weaker P_{TN} than MEGU. On the other hand, both Eraser and GDelete exhibit either a low P_{TN} or a low P_{TP}. We believe such performance disparity across different unlearning models is rooted in the underlying design of these unlearning algorithms. Specifically, as GDelete forces the node representations to align closely with randomly chosen nodes, the challenge edges always alter the predictions of the token nodes in the unlearning model. This leads to a low P_{TP} but a high P_{TN} of PANDA on GDelete. On the other hand, as Eraser partitions the graph into multiple shards, such change of graph structure makes the challenge edges fail to change the token nodes' prediction in the unlearning model, leading to a low P_{TN} but a high P_{TP} value on Eraser.

[4]https://github.com/MinChen00/Graph-Unlearning
[5]https://github.com/kunwu522/certified_edge_unlearning
[6]https://github.com/mims-harvard/GNNDelete
[7]https://github.com/xkLi-Allen/MEGU

Table 5: Robustness of PANDA against detection of adversarial perturbation.

Dataset	LP [44]	OD [44]	GGD [44]
CiteSeer	9.6%	3.6%	0.8%
LastFM	11.6%	0.8%	1.6%
CS	3.2%	0.4%	0.4%

6.5 Robustness of Verification (RQ_4)

Intuitively, the server can escape from verification by detecting the challenge edges. In this part of the experiments, we evaluate the robustness of PANDA by three existing graph adversarial perturbation detection methods: (1) LinkPred (**LP**) [44] flags the edges of low link prediction probability as adversarial edges; (2) OutlierDetect (**OD**) [44] detects adversarial edges based on the distribution of their neighborhood; (3) GraphGenDetect (**GGD**) [44] detects the adversarial edges by their subgraph structure. We measure the robustness of the challenge edges as the *detection ratio*, i.e., the percentage of the challenge edges that are correctly identified by the detection methods. For each dataset, we generate challenge edges that account for 1% of the total number of real edges.

Table 5 reports the robustness results of PANDA against the three detection methods. Overall, the detection ratio of PANDA's challenge edges remains consistently low, with 11.6% at maximum by LP, no more than 4% for OD and 2% for GGD, respectively. This demonstrates that the challenge edges are similar to the real edges in terms of their link prediction probability, neighborhood distribution, and subgraph structure. We provide additional analysis of these characteristics of challenge edges and real ones in Appendix E.7.

7 DISCUSSIONS

7.1 Verification of Removing Challenge Edges and Real Ones

So far, we have examined the verification of unlearning requests that consist solely of challenge edges. In practice, the unlearning requests can include both real and challenge edges. In this section, we analyze and evaluate the completeness verification guarantee of PANDA when the unlearning requests contain both real edges and challenge ones.

Table 6: Verification probability of PANDA when removing both challenge edges and real ones ($\alpha, \beta = 90\%$).

	Percentage of removed edges as challenge edges				
	30%	40%	50%	60%	70%
P_{TP}	93.6%	96.8%	98.4%	99.2%	99.2%
P_{TN}	92.8%	93.6%	94.8%	94.8%	97.2%

Deriving the verification probability becomes challenging, as it is difficult to quantify the likelihood that the token nodes' predictions change by removing both real edges and challenge ones. To simplify the reasoning, we assume that removing real edges does not affect the predictions of token nodes. This assumption holds when the real edges are at least L hops away from the token nodes, where L is the number of layers in the target model.

Formally, consider a set of t edges to be removed (i.e., $|E_U| = t$), among which $c \leq t$ are challenge edges. Let $E' \subset E_U$ denote the set of edges that a cheating server chooses to remove from the model, with $r = |E'|$. We analyze two cases — Case ❶: $r \geq c$ and Case ❷: $r < c$. For Case ❶, $\mathcal{S}$ can evade verification with the probability p_S if E' contains all challenge edges, or the probability $1 - p_N$ if E' contains only a subset of challenge edges. Here, p_S and p_N correspond to the soundness and necessity probabilities, respectively, as defined for the challenge-edge-only scenario (Sec. 4). For Case ❷, $\mathcal{S}$ can evade verification with probability $1 - p_N$ if E' contains at least one challenge edge. Therefore, the overall probability P that a cheating server can successfully evade verification by leveraging the prediction of a single token node is given by:

$$P = \begin{cases} \frac{\binom{t-c}{r-c}}{\binom{t}{r}} \cdot p_S + \frac{\binom{t-c}{r}}{\binom{t}{r}} + (1 - \frac{\binom{t-c}{r}}{\binom{t}{r}} - \frac{\binom{t-c}{r}}{\binom{t}{r}})(1 - p_N), & r \geq c \\ \frac{\binom{t-c}{r}}{\binom{t}{r}} + (1 - \frac{\binom{t-c}{r}}{\binom{t}{r}})(1 - p_N), & r < c. \end{cases} \tag{9}$$

Intuitively, a higher $\frac{c}{t}$ value (i.e., challenge edges constitute a larger portion of the unlearning request) results in a lower P value. Then with m token nodes, the *true positive probability* P_{TP} of verification is given by: $P_{TP} = 1 - P^m$. And the *true negative probability* P_{TN} is the same as the case with challenge edges only (Eqn. (2)): $P_{TN} = (p_S)^m$, as we assume removing real edges does not change the token nodes' predictions, The measurement of p_N and p_S is conducted in the same manner as for challenge edges only (Section 5.4). Notably, As a higher $\frac{c}{t}$ ratio leads to a lower P, it consequently results in a higher P_{TP}. When $c = t$ (i.e., the unlearning requests only consist of challenge edges), $P_{TP} = 1 - (1 - p_N)^m$ (i.e., Eqn. (1)). On the other hand, The number of token nodes that satisfy (α, β)−completeness then can be computed from P_{TP} and P_{TN}.

We evaluate the performance of PANDA when the unlearning requests consist of both challenge edges and real ones. For each trial, we pick 40 edges for removal. Among these, the challenge edges constitute 30%, 40%, 50%, 60%, 70% of the total. Across all settings, we fix $\alpha, \beta = 0.9$ and set the incompleteness ratio $s = 20\%$. Table 6 reports P_{TP} and P_{TN} with varying amounts of challenge edges in the unlearning requests. Overall, PANDA satisfies (α, β)-completeness across all the settings. Furthermore, both P_{TP} and P_{TN} increase when the amounts of challenge edges grows. This demonstrates the effectiveness of PANDA even when the removed edges consist of both challenge edges and real ones.

Property	CiteSeer		LastFM		CS	
	Ch.	Real	Ch.	Real	Ch	Real
Degree	10.5 ±3.9	11.7 ±2.3	23.3 ±8.6	27.0 ±12.3	18.1 ±5.4	22.8 ±14.3
Betweenness	.0029 ±.003	.0011 ±.002	.00039 ±.0004	.00026 ±.0004	.00007 ±.00005	.00007 ±.00009
Connectivity	2.32 ±1.3	2.88 ±2.2	5.64 ±4.5	6.92 ±4.6	4.16 ±2.0	6.59 ±5.4

Table 7: Challenge edges vs. real ones by their edge degree, edge betweenness, and edge connectivity.

7.2 Robustness of PANDA against Adaptive Attacks

In this section, we consider a malicious server that has partial knowledge of the verification mechanism. Specifically, it is aware that challenge edges can alter the structure of token nodes, thereby affecting their predictions. To exploit this, we focus on three structural properties of edges, namely *edge degree*, *edge betweenness*, and *edge connectivity*, and design adaptive attacks aimed at identifying challenge edges based on these properties.

To evaluate the resilience of the challenge edges against the property-based detection, we construct 100 challenge edges and compare them against 100 real edges randomly sampled from the dataset. Table 7 presents the results across three datasets, evaluating the edges based on degree, betweenness, and connectivity. The key observation is that challenge edges exhibit similar structural properties to real edges, indicating that the perturbations introduced by PANDA do not create easily detectable topological anomalies.

7.3 Adapt to Node and Feature Unlearning

While PANDA is designed for verifying the completeness of edge unlearning, it can be naturally extended to support node- and feature-level unlearning. Specifically, like edge unlearning, PANDA first selects a set of token nodes from. For node unlearning, it injects fake nodes and edges to $\mathcal{G}$; for feature unlearning, it adds fake features to the token nodes in $\mathcal{G}$. These modifications are crafted to alter the token nodes' predictions. Verification is then performed by requesting their removal of the injected elements and checking whether the token nodes' predictions revert to their original state.

8 CONCLUSION

In this paper, we introduce PANDA, the first probabilistic verification framework for incomplete graph unlearning. The key idea of PANDA is to identify a set of 1PF nodes as token nodes and to generate adversarial perturbations of these nodes as the challenge edges for verification. Our experiments demonstrate the effectiveness of PANDA and its superiority over the existing graph adversarial perturbation methods.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their feedback. This work was supported by the National Science Foundation (CNS-2029038; CNS-2135988). Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agency.

REFERENCES

[1] Faiz A Al-Khayyal and James E Falk. 1983. Jointly constrained biconvex programming. *Mathematics of Operations Research* 8, 2 (1983), 273–286.

[2] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine unlearning. In *Symposium on Security and Privacy (SP)*. IEEE, 141–159.

[3] Heng Chang, Yu Rong, Tingyang Xu, Wenbing Huang, Honglei Zhang, Peng Cui, Wenwu Zhu, and Junzhou Huang. 2020. A restricted black-box adversarial framework towards attacking graph embedding models. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 3389–3396.

[4] Jinyin Chen, Yangyang Wu, Xuanheng Xu, Yixian Chen, Haibin Zheng, and Qi Xuan. 2018. Fast gradient attack on network embedding. *arXiv preprint arXiv:1809.02797* (2018).

[5] Liang Chen, Jintang Li, Jiaying Peng, Tao Xie, Zengxu Cao, Kun Xu, Xiangnan He, Zibin Zheng, and Bingzhe Wu. 2020. A survey of adversarial learning on graphs. *arXiv preprint arXiv:2003.05730* (2020).

[6] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. 2022. Graph unlearning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 499–513.

[7] Jiali Cheng, George Dasoulas, Huan He, Chirag Agarwal, and Marinka Zitnik. 2023. GNNDelete: A General Strategy for Unlearning in Graph Neural Networks. In *International Conference on Learning Representations (ICLR)*.

[8] Eli Chien, Chao Pan, and Olgica Milenkovic. 2022. Certified graph unlearning. *arXiv preprint arXiv:2206.09140* (2022).

[9] Weilin Cong and Mehrdad Mahdavi. 2023. GraphEditor: An Efficient Graph Representation Learning and Unlearning Approach.

[10] Amirata Ghorbani and James Zou. 2019. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*. 2242–2251.

[11] Laura Graves, Vineel Nagisetty, and Vijay Ganesh. 2021. Amnesiac machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11516–11524.

[12] Yu Guo, Yu Zhao, Saihui Hou, Cong Wang, and Xiaohua Jia. 2023. Verifying in the dark: Verifiable machine unlearning by using invisible backdoor triggers. *IEEE Transactions on Information Forensics and Security* 19 (2023), 708–721.

[13] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1025–1035.

[14] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. 2021. Stealing links from graph neural networks. In *30th USENIX Security Symposium (USENIX Security)*. 2669–2686.

[15] Wei Jin, Yaxing Li, Han Xu, Yiqi Wang, Shuiwang Ji, Charu Aggarwal, and Jiliang Tang. 2021. Adversarial Attacks and Defenses on Graphs. *SIGKDD Explor. Newsl.* 22, 2 (Jan. 2021), 19–34.

[16] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[17] Chanhee Kwak, Junyeong Lee, Kyuhong Park, and Heeseok Lee. 2017. Let Machines Unlearn—Machine Unlearning and the Right to be Forgotten. In *Proceedings of the 23rd Americas Conference on Information Systems (AMCIS)*. 2253–2257.

[18] Xunkai Li, Yulin Zhao, Zhengyu Wu, Wentao Zhang, Rong-Hua Li, and Guoren Wang. 2024. Towards Effective and General Graph Unlearning via Mutual Evolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 13682–13690.

[19] Stuart L Pardau. 2018. The California consumer privacy act: Towards a European-style privacy regime in the United States. *J. Tech. L. & Pol'y* 23 (2018), 68.

[20] Protection Regulation. 2018. General data protection regulation. *Intouch* 25 (2018), 1–5.

[21] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz. 2015. Mlaas: Machine learning as a service. In *IEEE 14th international conference on machine learning and applications (ICMLA)*. 896–902.

[22] Benedek Rozemberczki and Rik Sarkar. 2020. Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models. In *Proceedings of the 29th ACM international conference on information & knowledge management*. 1325–1334.

[23] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.

[24] Ozan Sener and Silvio Savarese. 2017. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489* (2017).

[25] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[26] David Marco Sommer, Liwei Song, Sameer Wagh, and Prateek Mittal. 2022. Athena: Probabilistic Verification of Machine Unlearning. *Proceedings on Privacy Enhancing Technologies* 3 (2022), 268–290.

[27] Lichao Sun, Yingtong Dou, Carl Yang, Kai Zhang, Ji Wang, Philip S Yu, Lifang He, and Bo Li. 2022. Adversarial attack and defense on graph data: A survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 8 (2022), 7693–7711.

[28] Tsubasa Takahashi. 2019. Indirect adversarial attacks via poisoning neighbors for graph convolutional networks. In *2019 IEEE International Conference on Big Data (Big Data)*. 1395–1400.

[29] Jiajun Tan, Fei Sun, Ruichen Qiu, Du Su, and Huawei Shen. 2024. Unlink to unlearn: Simplifying edge unlearning in gnns. In *Companion Proceedings of the ACM on Web Conference 2024*. 489–492.

[30] Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. 2022. On the necessity of auditable algorithmic definitions for machine unlearning. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, USA, 4007–4022.

[31] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. 2017. Graph attention networks. *stat* 1050, 20 (2017), 10–48550.

[32] Binghui Wang and Neil Zhenqiang Gong. 2019. Attacking graph-based classification via manipulating the graph structure. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, London, United Kingdom, 2023–2040.

[33] Cheng-Long Wang, Mengdi Huai, and Di Wang. 2023. Inductive graph unlearning. In *32nd USENIX Security Symposium (USENIX Security)*. 3205–3222.

[34] Jiasi Weng, Shenglong Yao, Yuefeng Du, Junjie Huang, Jian Weng, and Cong Wang. 2024. Proof of unlearning: Definitions and instantiation. *IEEE Transactions on Information Forensics and Security* 19 (2024), 3309–3323.

[35] Eric Wong and Zico Kolter. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International conference on machine learning*. 5286–5295.

[36] Fan Wu, Yunhui Long, Ce Zhang, and Bo Li. 2022. Linkteller: Recovering private edges from graph neural networks via influence analysis. In *2022 ieee symposium on security and privacy (S&P)*. 2005–2024.

[37] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. 2019. Adversarial Examples for Graph Data: Deep Insights into Attack and Defense. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. 4816–4823.

[38] Jiancan Wu, Yi Yang, Yuchun Qian, Yongduo Sui, Xiang Wang, and Xiangnan He. 2023. Gif: A general graph unlearning strategy via influence function. In *Proceedings of the ACM Web Conference 2023*. 651–661.

[39] Kun Wu, Jie Shen, Yue Ning, Ting Wang, and Wendy Hui Wang. 2023. Certified Edge Unlearning for Graph Neural Networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2606–2617.

[40] Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. 2023. Graph neural networks for natural language processing: A survey. *Foundations and Trends® in Machine Learning* 16, 2 (2023), 119–328.

[41] Heng Xu, Tianqing Zhu, Lefeng Zhang, and Wanlei Zhou. 2024. Really Unlearned? Verifying Machine Unlearning via Influential Sample Pairs. *arXiv preprint arXiv:2406.10953* (2024).

[42] Kaidi Xu, Hongge Chen, Sijia Liu, Pin Yu Chen, Tsui Wei Weng, Mingyi Hong, and Xue Lin. 2019. Topology attack and defense for graph neural networks: An optimization perspective. In *28th International Joint Conference on Artificial Intelligence, IJCAI 2019*. International Joint Conferences on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Macau, China, 3961–3967.

[43] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[44] Xiaojun Xu, Hanzhang Wang, Alok Lal, Carl A Gunter, and Bo Li. 2023. Edog: Adversarial edge detection for graph neural networks. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. 291–305.

[45] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*. 40–48.

[46] Yuanshun Yao, Zhujun Xiao, Bolun Wang, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. 2017. Complexity vs. performance: empirical analysis of machine learning as a service. In *Proceedings of the 2017 Internet Measurement Conference*. 384–397.

[47] Binchi Zhang, Zihan Chen, Cong Shen, and Jundong Li. 2024. Verification of machine unlearning is fragile. In *Proceedings of the 41st International Conference on Machine Learning* (Vienna, Austria) *(ICML)*. JMLR.org, Vienna, Austria, Article 2422, 22 pages.

[48] Jiahao Zhang. 2024. Graph unlearning with efficient partial retraining. In *Companion Proceedings of the ACM on Web Conference 2024*. 1218–1221.

[49] Zehong Zhang, Lifan Chen, Feisheng Zhong, Dingyan Wang, Jiaxin Jiang, Sulin Zhang, Hualiang Jiang, Mingyue Zheng, and Xutong Li. 2022. Graph neural network approaches for drug-target interactions. *Current Opinion in Structural Biology* 73 (2022), 102327.

[50] Wenyue Zheng, Ximeng Liu, Yuyang Wang, and Xuanwei Lin. 2023. Graph Unlearning Using Knowledge Distillation. In *International Conference on Information and Communications Security*. 485–501.

[51] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD*

international conference on knowledge discovery & data mining. 2847–2856.
[52] Daniel Zügner and Stephan Günnemann. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *International Conference on Learning Representations (ICLR)*.
[53] Daniel Zügner and Stephan Günnemann. 2019. Certifiable Robustness and Robust Training for Graph Convolutional Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 246–256.
[54] Daniel Zügner and Stephan Günnemann. 2020. Certifiable robustness of graph convolutional networks under structure perturbations. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 1656–1665.

APPENDIX

A BOUND DERIVATION

In this section, we derive the bounds that are defined in Section 5.2 (Equations 6 and 7). Without losing generality, we assume the perturbation is added as a new edge between two nodes v_a and v_b for the derivation of all the bounds in the following discussions.

Element-wise bounds. Every element of $\mathring{\tilde{\mathbf{A}}}$ can be bounded as $L_{ij} \leq \mathring{\tilde{\mathbf{A}}}_{ij} \leq U_{ij}$. There are four cases regarding each entry of $\mathring{\tilde{\mathbf{A}}}$:

$$\mathring{\tilde{\mathbf{A}}}_{ij} = \begin{cases} \frac{1}{\sqrt{d_i+1}\sqrt{d_j+1}}, & \text{if } i = a \wedge j = b \text{ (Case I)} \\ \frac{1}{\sqrt{d_i+1}\sqrt{d_j}}, & \text{if } i = a \wedge j \neq b \text{ (Case II)} \\ \frac{1}{\sqrt{d_i}\sqrt{d_j+1}}, & \text{if } i \neq a \wedge j = b \text{ (Case III)} \\ \mathring{\mathbf{A}}_{ij}, & \text{Otherwise (Case IV)}. \end{cases} \tag{10}$$

Among the four cases, as the value of Case IV is larger than the other three cases if $\mathring{\mathbf{A}}_{ij} \neq 0$, the upper bound of each element in $\mathring{\tilde{\mathbf{A}}}$ is derived as $U_{ij} = \max(\mathring{\mathbf{A}}_{ij}, \frac{1}{\sqrt{d_i+1}\sqrt{d_j+1}})$.

The lower bound of each element is $L_{ij} = \min\{\mathring{\mathbf{A}}_{ij}, \frac{1}{\sqrt{d_i+1}\sqrt{d_j+1}}\}$ for $i \neq j$, where the first term covers the case when $\mathring{\mathbf{A}}_{ij} = 0$, and the second term covers Case I) which is the smallest among all the cases. When $i = j$, $L_{ij} = \frac{1}{d_i+1}$.

Row-wise bounds. We have the following six cases of computing the row sum $\mathring{\tilde{\mathbf{A}}}_i$ for each node i. We use $\mathcal{N}_i$ to denote the neighborhood of v_i in the original graph, including v_i itself. Recall that we assume the adversarial edge is added between the two nodes v_a and v_b.

I. $i = a$ (i.e., the row of the node v_a):

$$\mathring{\tilde{\mathbf{A}}}_i = \frac{1}{\sqrt{d_i+1}} \sum_{\substack{j \neq i \\ j \in \mathcal{N}_i}} \frac{1}{\sqrt{d_j}} + \frac{1}{\sqrt{d_i+1}\sqrt{d_b+1}} + \frac{1}{d_i+1} \tag{11}$$

II. $i = b$ (i.e., the row of the node v_b):

$$\mathring{\tilde{\mathbf{A}}}_i = \frac{1}{\sqrt{d_i+1}} \sum_{\substack{j \neq i \\ j \in \mathcal{N}_i}} \frac{1}{\sqrt{d_j}} + \frac{1}{\sqrt{d_i+1}\sqrt{d_a+1}} + \frac{1}{d_i+1} \tag{12}$$

III. $i \neq a \wedge i \neq b \wedge a \in \mathcal{N}_i \wedge b \in \mathcal{N}_i$ (i.e., the row of node v which is neither v_a nor v_b but in the neighborhood of both v_a and v_b):

$$\mathring{\tilde{\mathbf{A}}}_i = \frac{1}{\sqrt{d_i}} \sum_{\substack{j \neq i \\ j \in \mathcal{N}_i \setminus \{a,b\}}} \frac{1}{\sqrt{d_j}} + \frac{1}{\sqrt{d_i}\sqrt{d_a+1}} + \frac{1}{\sqrt{d_i}\sqrt{d_b+1}} + \frac{1}{d_i} \tag{13}$$

IV. $i \neq a \wedge i \neq b \wedge a \in \mathcal{N}_i \wedge b \notin \mathcal{N}_i$ (i.e., the row of node v which is neither v_a nor v_b but in v_a's neighborhood only):

$$\mathring{\tilde{\mathbf{A}}}_i = \frac{1}{\sqrt{d_i}} \sum_{\substack{j \neq i \\ j \in \mathcal{N}_i \setminus \{a\}}} \frac{1}{\sqrt{d_j}} + \frac{1}{\sqrt{d_i}\sqrt{d_a+1}} + \frac{1}{d_i} \tag{14}$$

V. $i \neq a \wedge i \neq b \wedge a \notin \mathcal{N}_i \wedge b \in \mathcal{N}_i$ (i.e., the row of node v which is neither v_a nor v_b but in v_b's neighborhood only):

$$\mathring{\tilde{\mathbf{A}}}_i = \frac{1}{\sqrt{d_i}} \sum_{\substack{j \neq i \\ j \in \mathcal{N}_i \setminus \{b\}}} \frac{1}{\sqrt{d_j}} + \frac{1}{\sqrt{d_i}\sqrt{d_b+1}} + \frac{1}{d_i} \tag{15}$$

VI. $i \neq a \wedge i \neq b \wedge a \notin \mathcal{N}_i \wedge b \notin \mathcal{N}_i$ (i.e., the row of node v which is neither v_a nor v_b nor in the neighborhood of v_a and v_b):

$$\sum_j \mathring{\tilde{\mathbf{A}}}_{ij} = \frac{1}{\sqrt{d_i}} \sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{d_j}} + \frac{1}{d_i} \tag{16}$$

It can be easily observed that $\mathring{\tilde{\mathbf{A}}}_i$ of Cases IV - VI is larger than that of Cases I III. Therefore, the row-wise lower bound L_i^{row} should be the minimum value between Case I, Case II, and Case III. In particular, $\mathring{\tilde{\mathbf{A}}}_i$ of these three cases reaches the lowest when $d_a = d_b = d_{max}$, where d_{max} is the highest node degree in the original graph. Therefore, we derive the row-wise lower bound L_i^{row} as follows:

$$\begin{aligned} L_i^{row} = \min\Big(&\frac{1}{\sqrt{d_i+1}} \sum_{\substack{j \neq i \\ j \in \mathcal{N}_i}} \frac{1}{\sqrt{d_j}} + \frac{1}{\sqrt{d_i+1}\sqrt{d_{max}+1}} + \frac{1}{d_i+1}, \\ &\frac{1}{\sqrt{d_i}} \sum_{\substack{j \neq i \\ j \in \mathcal{N}_i \setminus \{a,b\}}} \frac{1}{\sqrt{d_j}} + \frac{2}{\sqrt{d_i}\sqrt{d_{max}+1}} + \frac{1}{d_i}\Big), \end{aligned} \tag{17}$$

Similarly, the upper bound U_i^{row} of $\mathring{\tilde{\mathbf{A}}}_i$ should be the maximum value of $\mathring{\tilde{\mathbf{A}}}_i$ of Case I and Case VI:

$$\begin{aligned} U_i^{row} = \max\Big(&\frac{1}{\sqrt{d_i+1}} \sum_{\substack{j \neq i \\ j \in \mathcal{N}_i}} \frac{1}{\sqrt{d_j}} + \frac{1}{\sqrt{d_i+1}\sqrt{d_{min}+1}} + \frac{1}{d_i+1}, \\ &\frac{1}{\sqrt{d_i}} \sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{d_j}} + \frac{1}{d_i}\Big), \end{aligned} \tag{18}$$

where d_{min} is the minimum degree of all the nodes in the original graph.

Global bounds. To compute the lower and upper bound of $|\mathring{\tilde{\mathbf{A}}} - \mathring{\mathbf{A}}|$, we first consider the element-wise distance, i.e., $|\mathring{\tilde{\mathbf{A}}}_{ij} - \mathring{\mathbf{A}}_{ij}|$ for all i, j. We have the following five cases:

$$|\mathring{\tilde{\mathbf{A}}}_{ij} - \mathring{\mathbf{A}}_{ij}| = \begin{cases} \left|\frac{1}{\sqrt{d_i+1}\sqrt{d_j+1}} - \frac{1}{\sqrt{d_i}\sqrt{d_j}}\right|, & i = a \wedge j = b \text{ (Case I)} \\ \left|\frac{1}{\sqrt{d_i+1}\sqrt{d_j}} - \frac{1}{\sqrt{d_i}\sqrt{d_j}}\right|, & i = a \wedge j \neq b \wedge i \neq j \text{ (Case II)} \\ \left|\frac{1}{\sqrt{d_i}\sqrt{d_j+1}} - \frac{1}{\sqrt{d_i}\sqrt{d_j}}\right|, & i \neq a \wedge j = b \wedge i \neq j \text{(Case III)} \\ \left|\frac{1}{d_i+1} - \frac{1}{d_i}\right|, & i = j \wedge (i = a \vee i = b) \text{ (Case IV)} \\ 0, & i \neq a \wedge j \neq b \text{ (Case V)} \end{cases} . \tag{19}$$

Based on the element-wise distance, the matrix-wise distance between $\mathring{\hat{\mathbf{A}}}$ and $\mathring{\mathbf{A}}$ is computed as follows:

$$\begin{aligned}|\mathring{\hat{\mathbf{A}}} - \mathring{\mathbf{A}}| &= \sum_{i,j} |\mathring{\hat{\mathbf{A}}}_{ij} - \mathring{\mathbf{A}}_{ij}| \\ &= 2\Big(\frac{1}{\sqrt{d_a}\sqrt{d_b}} - \frac{1}{\sqrt{d_a+1}\sqrt{d_b+1}}\Big) + \Big(\frac{1}{d_a} - \frac{1}{d_a+1}\Big) + \Big(\frac{1}{d_b} - \frac{1}{d_b+1}\Big) \\ &\quad + \sum_{\substack{j\neq a \\ j\in\mathcal{N}_a}} \frac{1}{\sqrt{d_j}}\Big(\frac{1}{\sqrt{d_a}} - \frac{1}{\sqrt{d_a+1}}\Big) + \sum_{\substack{j\neq b \\ j\in\mathcal{N}_b}} \frac{1}{\sqrt{d_j}}\Big(\frac{1}{\sqrt{d_b}} - \frac{1}{\sqrt{d_b+1}}\Big).\end{aligned} \tag{20}$$

Since $d_a, d_b \geq 1$, Eqn (20) is monotonically decreasing w.r.t d_a and d_b. Thus the global upper bound U_{glob} is achieved when $d_a = d_b = d_{min}$:

$$\begin{aligned}U_{glob} &= 2\Big(\frac{1}{\sqrt{d_{min}}\sqrt{d_{min}}} - \frac{1}{\sqrt{d_{min}+1}\sqrt{d_{min}+1}}\Big) \\ &\quad + \Big(\frac{1}{d_{min}} - \frac{1}{d_{min}+1}\Big) + \Big(\frac{1}{d_{min}} - \frac{1}{d_{min}+1}\Big) \\ &\quad + 2\sum_{\substack{j\notin\{a,b\} \\ j\in\mathcal{N}_{min}}} \frac{1}{\sqrt{d_j}}\Big(\frac{1}{\sqrt{d_{min}}} - \frac{1}{\sqrt{d_{min}+1}}\Big) \\ &= 4\Big(\frac{1}{d_{min}} - \frac{1}{d_{min}+1}\Big) + 2\sum_{\substack{j\notin\{a,b\} \\ j\in\mathcal{N}_{min}}} \frac{1}{\sqrt{d_j}}\Big(\frac{1}{\sqrt{d_{min}}} - \frac{1}{\sqrt{d_{min}+1}}\Big)\end{aligned} \tag{21}$$

where $\mathcal{N}_{min}$ denotes the neighborhood of the node that has the lowest node degree. We do not consider the lower global bound as its inclusion would make the optimization problem non-convex.

Next, we derive the lower and upper bounds of the number of "positive" entries in $\mathring{\hat{\mathbf{A}}} - \mathring{\mathbf{A}}$ for the edge-addition constraint $\tilde{\mathbf{A}}_{ij} \geq \mathbf{A}_{ij}, \forall 1 \leq i, j \leq n$ in the optimization problem (Eqn. (4)). As the perturbation is added between the node pair (v_a, v_b), it must be true that $\mathbf{A}_{ab} = 0$. Thus the positive change on the entry $\mathring{\mathbf{A}}_{ab}$ is $\frac{2}{\sqrt{d_a+1}\sqrt{d_b+1}}$. Hence, the lower bound of the number of "positive" entries in $\mathring{\hat{\mathbf{A}}} - \mathring{\mathbf{A}}$ (denoted as L_{pos}) is achieved when $d_a = d_b = d_{max}$:

$$L_{pos} = \frac{2}{d_{max}+1}. \tag{22}$$

Similarly, the upper bound of the number of "positive" entries in $\mathring{\hat{\mathbf{A}}} - \mathring{\mathbf{A}}$ (denoted as L_{pos}) is achieved when $d_a = d_b = d_{min}$:

$$U_{pos} = \frac{2}{d_{min}+1}. \tag{23}$$

B PROOFS

B.1 Proof of Theorem 5.2

Proof. To prove PANDA can provide a perfect verification guarantee, we first prove that the challenge edges provide perfect soundness probability (i.e., $p_S = 100\%$) and perfect necessity (i.e., $p_N = 100\%$). In other words, the following must hold:

- *Soundness*: $y_v^{E_{\mathcal{A}}} \neq y_v$, where $y^{E_{\mathcal{A}}}$ and y_v are v's label predicted by f trained on $\mathcal{G}(V, E \cup E_{\mathcal{A}})$ and $\mathcal{G}(V, E)$, respectively.
- *Necessity*: $y_v^{E_{\mathcal{A}}} = y_v^{E'}, \forall E' \subset E_{\mathcal{A}}$, where $y_v^{E_{\mathcal{A}}}$ and $y_v^{E'}$ are v's label predicted by f trained on $G(V, E \cup E_{\mathcal{A}})$ and $G(V, E \cup E')$, respectively.

Soundness. Given a graph $\mathcal{G}$ and any 1PF node $v \in \mathcal{G}$, let $E_{\mathcal{A}} = \{e_1, \ldots, e_k\}$ be a set of consistent 1-edge perturbations of v, where $k \geq 1$ can be any arbitrary number. Let $y^{E_{\mathcal{A}}}$ and y_v be v's label predicted by the GCN on $\mathcal{G}(V, E \cup E_{\mathcal{A}})$ and $\mathcal{G}(V, E)$, respectively. To prove that $y_v^{E_{\mathcal{A}}} \neq y_v$, we will prove that, for any 1-layer GCN with a binary classification task, the following must hold:

$$y_v^{E_{\mathcal{A}}} = y_v^{e_i}, \forall e_i \in E_{\mathcal{A}}, \tag{24}$$

where $y_v^{e_i}$ is v's label predicted by the GCN on $\mathcal{G}(V, E \cup E_{e_i})$.

For a 1-layer GCN model, the logits of a given token node $v \in V$ can be represented as

$$Z_v = \theta^\intercal \sum_{u\in\mathcal{N}_v} \frac{1}{\sqrt{d_u}\sqrt{d_v}}\mathbf{X}_u, \tag{25}$$

where θ, $\mathcal{N}_v$, d_u and x_u denote the parameters of the GCN model, the neighbors of node v including v itself, the degree of node u, and the node features of u, respectively. Then, the label y of v predicted by a GNN model θ can be formalized as

$$y := \arg\max Z_v. \tag{26}$$

We further use $Z_{v,y}$ to denote the logit of label y of v predicted by the model trained on $\mathcal{G}(V, E)$.

Given a set of 1-perturbations of v, $E_{\mathcal{A}} = \{e_1, \ldots, e_k\}$, each e_i taking the format of (v, p_i). The logit of node v predicted from the graph perturbed by a single 1-perturbation e is:

$$Z_v^{e_i} = \theta^\intercal\Big(\sum_{u\in\mathcal{N}(v)} \frac{1}{\sqrt{d_u}\sqrt{d_v+1}}\mathbf{X}_u + \frac{1}{\sqrt{d_{p_i}}\sqrt{d_v+1}}\mathbf{X}_{p_i}\Big) \tag{27}$$

For any 1-perturbation edge $e_i \in E_{\mathcal{A}}$, WLOG let y' be the perturbed prediction of v over $\mathcal{G}(V, E \cup e_i)$. Since y' is the predicted label of node v on $\mathcal{G}(V, E \cup \{e_i\})$, we have

$$Z_{v,y'}^{e_i} > Z_{v,y}^{e_i}, \tag{28}$$

Next, we aim to derive that $Z_{v,y'}^{E_{\mathcal{A}}} > Z_{v,y}^{E_{\mathcal{A}}}$, for any label $y \neq y'$. In other words, y' will also be v's label predicted on $\mathcal{G}(V, E \cup E_{\mathcal{A}})$.

First, by following Eqn (27), we can easily derive the logit of y' for the prediction over $\mathcal{G}(V, E \cup \{e_i\})$ as follows:

$$Z_{v,y'}^{e_i} = \theta_{y'}^\intercal\Big(\sum_{u\in\mathcal{N}(v)} \frac{1}{\sqrt{d_u}\sqrt{d_v+1}}\mathbf{X}_u + \frac{1}{\sqrt{d_{p_i}+1}\sqrt{d_v+1}}\mathbf{X}_{p_i}\Big). \tag{29}$$

Similarly, the logit of y' for the prediction over $\mathcal{G}(V, E \cup E_{\mathcal{A}})$ is computed as follows:

$$Z_{v,y'}^{E_{\mathcal{A}}} = \theta_{y'}^\intercal\Big(\sum_{u\in\mathcal{N}(v)} \frac{1}{\sqrt{d_u}\sqrt{d_v+1}} + \sum_{i=1}^{k} \frac{1}{\sqrt{d_{p_i}+1}\sqrt{d_v+1}}\Big). \tag{30}$$

As y' is the predicted label on $\mathcal{G}(V, E \cup \{e_i\})$, we have the following holding for the label $y \neq y'$:

$$\sum_{i=1}^{k} Z_{v,y'}^{e_i} > \sum_{i=1}^{k} Z_{v,y}^{e_i} \tag{31}$$

With Eqn. (29), we can expand the left-hand side of Eqn. (31) as follows (for 1-layer GCNs):

$$\begin{aligned}\sum_{i=1}^{k} Z_{v,y'}^{e_i} &= \theta_{y'}^\intercal\Big(k\sum_{u\in\mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}} + \sum_{i=1}^{k} \frac{\mathbf{X}_{p_i}}{\sqrt{d_{p_i}+1}\sqrt{d_v+1}}\Big) \\ &= \theta_{y'}^\intercal\Big((k-1)\sum_{u\in\mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}} + \sum_{u\in\mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}} + \sum_{i=1}^{k} \frac{\mathbf{X}_{p_i}}{\sqrt{d_{p_i}+1}\sqrt{d_v+1}}\Big) \\ &= (k-1)\theta_{y'}^\intercal \sum_{u\in\mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}} + \theta_{y'}^\intercal\Big(\sum_{u\in\mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}} + \sum_{i=1}^{k} \frac{\mathbf{X}_{p_i}}{\sqrt{d_{p_i}+1}\sqrt{d_v+1}}\Big) \\ &= (k-1)\theta_{y'}^\intercal \sum_{u\in\mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}} + Z_{v,y'}^{E_{\mathcal{A}}}.\end{aligned} \tag{32}$$

Algorithm 1: 1PF algorithm

Input: Surrogate model f_S, graph $\mathcal{G}$, # of iterations T
Output: A 1PF node $v^\star \in \mathcal{G}$, a 1-edge perturbation $e^\star$, and the attacking label $y^\star$

Function FindNode($f_S, \mathcal{G}, T$):
  $v^\star \leftarrow$ NULL, $e^\star \leftarrow$ NULL
  while $e^\star$=*NULL and* $\exists$ *an unpicked node in* $\mathcal{G}$ **do**
    $v \leftarrow$ an unpicked node randomly chosen from $\mathcal{G}$
    $y^\star \leftarrow$ the second-best label of v
    $e^\star \leftarrow$ FindAdvE ($f_S, \mathcal{G}, v, y^\star, T$)
    Mark v as picked
    if $e^\star$ = *NULL* **then**
      Continue
    else
      $v^\star = v$
      Break
    end
  end
  return $v^\star, y^\star$
Function FindAdvE($f_S, \mathcal{G}, v, y^\star, T$):
  $i \leftarrow 1$, $e^\star \leftarrow$ NULL
  while $i \leq T$ **do**
    Randomly choose a node $v' \in \mathcal{G}$ where $y_{v'} = y^\star$ and $edge(v, v') \notin \mathcal{G}$
    Generate a sub-adjacency matrix $\mathbf{A}^v$ that only includes the L-hop neighborhood of v and v'
    $e^\star \leftarrow$ the approximate solution of the joint constrained bilinear program by searching in the space $\Phi(\mathbf{A}^v)$ (Eqn. (8))
    if $e^\star$=*NULL* **then**
      $i \leftarrow i + 1$
    else
      Break;
    end
  end
  return $e^\star$

Algorithm 2: Preparation phase of PANDA

Input: Graph $\mathcal{G}(V, E)$, verification guarantees α and β, # of challenge edges B, # of iterations T
Output: A set of token nodes $V^\star$ and their challenge edges $E^\star$

$m \leftarrow$ # of node tokens to satisfy (α, β)-completeness guarantee // Eqn. (3)
$f_S \leftarrow$ A trained surrogate model
$V^\star, E^\star \leftarrow \emptyset$
while $|V^\star| < m$ **do**
  $v^\star, y^\star \leftarrow$ FindNode ($f_S, \mathcal{G}, T$) // Algorithm 1
  $E_v \leftarrow \emptyset$
  while $|E_v| < B/m$ **do**
    $e^\star \leftarrow$FindAdvE ($f_S, \mathcal{G}, v^\star, y^\star, T$) // Algorithm 1
    if $e^\star \neq$*NULL* **then**
      $E_v \leftarrow E_v \cup \{e^\star\}$;
    end
  end
  $V^\star \leftarrow V^\star \cup \{v^\star\}$
  $E^\star \leftarrow E^\star \cup \{E_v\}$
end
return $V^\star, E^\star$

Similarly, the right-hand side of Eqn. (31) can be expanded as

$$\sum_{i=1}^{k} Z_{v,y}^{e_i} = (k-1)\theta_y^\intercal \sum_{u \in \mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}} + Z_{v,y}^{E_{\mathcal{A}}}. \tag{33}$$

With the expansions on both sides, Eqn. (31) can be rewritten as:

$$(k-1)\theta_{y'}^\intercal \textstyle\sum_{u \in \mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}} + Z_{v,y'}^{E_{\mathcal{A}}} > (k-1)\theta_y^\intercal \sum_{u \in \mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}} + Z_{v,y}^{E_{\mathcal{A}}}. \tag{34}$$

This can derive the following:

$$Z_{v,y'}^{E_{\mathcal{A}}} - Z_{v,y}^{E_{\mathcal{A}}} > (k-1)(\theta_y^\intercal - \theta_{y'}^\intercal) \sum_{u \in \mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}}. \tag{35}$$

The right-hand side of Eqn. (35) can be further derived as follows:

$$\begin{aligned}
&(k-1)(\theta_y^\intercal - \theta_{y'}^\intercal) \sum_{u \in \mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v+1}} \\
=&\frac{(k-1)}{\sqrt{d_v+1}} \Big(\theta_y^\intercal \sum_{u \in \mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}} - \theta_{y'}^\intercal \sum_{u \in \mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}} \Big) \\
=&\frac{(k-1)\sqrt{d_v}}{\sqrt{d_v+1}} \Big(\theta_y^\intercal \sum_{u \in \mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v}} - \theta_{y'}^\intercal \sum_{u \in \mathcal{N}_v} \frac{\mathbf{X}_u}{\sqrt{d_u}\sqrt{d_v}} \Big) \\
=&\frac{(k-1)\sqrt{d_v}}{\sqrt{d_v+1}} (Z_{v,y} - Z_{v,y'}),
\end{aligned} \tag{36}$$

Since y is the predicted label on the original graph, it must be true that $Z_{v,y} - Z_{v,y'} > 0$. Therefore, $Z_{v,y'}^{E_{\mathcal{A}}} - Z_{v,y}^{E_{\mathcal{A}}} > 0$. Consequently, y' will be the predicted label on $\mathcal{G}(V, E_{\mathcal{A}})$. Therefore, the challenge edges provide 100% soundness.

Necessity. Given a subset $E' \subset E_{\mathcal{A}}$, we can obtain another subset by removing E' from $E_{\mathcal{A}}$ as

$$E^* = E_{\mathcal{A}} \backslash E'. \tag{37}$$

By following Theorem 5.3, As $\forall e_i \in E^*$ is a 1-edge perturbation of v, we have

$$y = y', \forall e_i \in E^*, \tag{38}$$

where y and y' are the labels of v predicted by f trained on $G(V, E \cup E^*)$ and $G(V, E \cup \{e_i\})$, respectively. We also have:

$$y = y', \forall e_i \in E_{\mathcal{A}}, \tag{39}$$

where y and y' are the labels of v predicted by f trained on $G(V, E \cup E_{\mathcal{A}})$ and $G(V, E \cup \{e_i\})$, respectively. Following both Equations (38) and (39), we have:

$$y = y', \forall E' \in E_{\mathcal{A}}, \tag{40}$$

where y and y' are the labels of v predicted by f trained on $G(V, E \cup E_{\mathcal{A}})$ and $G(V, E \cup (E_{\mathcal{A}} \backslash E'))$, respectively. Therefore, the challenge edges provide 100% necessity. Similar to soundness,

Since both $p_S = p_N = 100\%$, they lead to $P_{TP} = 100\%$ and $P_{TN} = 100\%$ by following Eqn. (1) and (2), respectively. □

Table 8: Statistic of datasets

Dataset	# nodes	# edges	# features	# classes
CiteSeer	3,327	9,104	3,703	6
LastFM	7,624	55,612	128	18
CS	10,000	49,650	6,805	15

Table 9: Soundness and necessity probabilities (p_S, p_N) of challenge edges generated by PANDA and baselines. The best results are marked with green.

Method	CiteSeer		LastFM		CS	
	p_S	p_N	p_S	p_N	p_S	p_N
Nettack	87.6%	38.0%	84.8%	46.4%	83.0%	31.0%
SGAttack	66.8%	47.2%	75.6%	50.0%	72.0%	34.8%
FGA	83.8%	44.0%	72.8%	44.8%	79.6%	33.6%
PANDA	100%	78.8%	98.0%	54.2%	100%	60.8%

B.2 Proof of Theorem 5.3

Proof. We have demonstrated the soundness and necessity of a single 1PF token node in Theorem 5.2. Next, we extend this theorem to show that the soundness and necessity of multiple independent 1PF token nodes also hold. Given a set of token nodes $V^\star \in \mathcal{G}$, with $\mathcal{E} = \cup_{v \in V^\star} E_v$ representing the corresponding set of 1-edge perturbations for $V^\star$. We assume that $\forall v \in V^\star$ are independent and $\forall e \in \mathcal{E}$ are consistent. This ensures that the prediction change of any individual token node does not influence others, and no perturbation in $\mathcal{E}$ connects any two 1PF token nodes in $V^\star$. Since each $v \in V^\star$ and its corresponding set of 1-edge perturbations satisfy both soundness and necessity, the entire set $V^\star$ guarantees 100% soundness and 100% necessity. Similarly, with $p_S = p_N = 100\%$ for $V^\star$, this results in $P_{TP} = 100\%$ and $P_{TN} = 100\%$. □

C PSEUDO CODE

1PF Algorithm. Algorithm 1 (Appendix C) presents the pseudo-code of finding token nodes and challenge edges. It consists of two main functions: (1) the FindNode() function that finds a 1PF node from $\mathcal{G}$ as the token node, and (2) the FindAdvE() function that constructs the 1-edge perturbations of the token node. We explain these two functions briefly.

The FindNode() function is designed to identify a 1PF (1-perturbation feasible) node within a graph $\mathcal{G}$. It starts by randomly selecting a node v from $\mathcal{G}$ and choosing an attacking label $y^\star$ for v. The label $y^\star$ is selected as the second-best label for v, which is the label with the second-highest classification probability according to the surrogate model trained on the original graph. This choice is based on the intuition that it is easier to change v's prediction to $y^\star$ than the other labels. Next, the function calls FindAdvE() to construct a 1-edge perturbation that, when inserted into the graph, will change v's prediction to $y^\star$. If FindAdvE() fails to return a valid perturbation, FindNode() selects a new node and repeats the process. This iterative procedure continues until either a 1PF node is found or all nodes in $\mathcal{G}$ have been examined.

The FindAdvE() function is designed to construct an adversarial edge for the node v that can alter v's predictions to $y^\star$. Enumerating all possible edges and evaluating their impact on v's predictions is time-consuming, particularly for large graphs. To improve efficiency, FindAdvE() uses a heuristic to limit the search space. Since a node can only influence its neighbors, the function constructs a sub-adjacency matrix $\mathbf{A}^v$ that includes only the L-hop neighbors of v and v', where L is the number of layers in the GNN. This matrix $\mathbf{A}^v$ is significantly smaller than the full adjacency matrix $\mathbf{A}$, reducing the search complexity. The function then defines the search space $\Phi(\mathbf{A}^v)$ (Eqn (8)) and seeks a 1-edge perturbation using an approximate solution of the bound-and-search algorithm. This approach applies a linear relaxation of ReLU within the defined search space. FindAdvE() will either find a valid 1-edge perturbation or terminate after T iterations, where T is a user-defined parameter.

Preparation phase of PANDA. Algorithm 2 presents the pseudo-code of the preparation phase of PANDA. First, it calculates the number m of token nodes required to satisfy the (α, β)-completeness guarantee (Eqn. (3)). Then it trains a surrogate model f_S. Next, it generates m token nodes by calling the FindNode() function (Algorithm 1). For each token node, it constructs B/m challenge edges by calling the FindAdvE() function. These challenge edges are guaranteed to be consistent, as their insertion will change the token node's prediction to be the same label $y^\star$, which is one of the input parameters of the FindAdvE() function.

D STATISTICS OF THE DATASETS

Table 8 reports the statistics of the three datasets.

E ADDITIONAL EXPERIMENTAL RESULTS

E.1 Soundness and Completeness Probabilities of Challenge Edges

We compare the performance of PANDA with baseline methods in terms of the soundness probability (p_S) and necessity probability (p_N) of the challenge edges, with results detailed in Table 9. Overall, PANDA significantly outperforms all baselines in both p_S and p_N values. Specifically, while the p_S values for all baseline methods do not exceed 87.6%, PANDA achieves at least 98%, reaching up to 100% on the CiteSeer and CS datasets. A similar pattern is observed for the p_N values. For instance, with the CiteSeer dataset, the p_N values for baseline methods do not exceed 47.2%, whereas PANDA reaches 78.8%.

Due to these low p_N values, the baseline methods fail to determine a suitable m value (Eqn. (3)) for the number of token nodes. For instance, with $\alpha = \beta = 0.7$ and $p_S = 83.8\%, p_N = 44.0\%$ by FGA on the Citeseer dataset, we have the lowerbound of m as $\lceil \log_{1-p_N}(1-\alpha) \rceil = 3$ and the upperbound of m as $\lfloor \log_{p_S} \beta \rfloor = 2$. Since there is no overlapping valid range for m, it is not possible to identify a m value to satisfy the required (70%, 70%)-completeness.

E.2 Comparison with Link Membership Inference Attacks

Table 10 reports the results of P_{TP} and P_{TN} of both PANDA and StealLink, a SOTA LMIA. We observe that StealLink mostly fails to be an effective verification method, as it always flags all trials positive (i.e., at least one removed edge is inferred as a member), and thus consistently offers 100% P_{TP} and 0% P_{TN}. The failure of StealLink comes from the fact that it is difficult to completely

Table 10: Verification accuracy of PANDA and LMIA.

Method	Meas.	CiteSeer	LastFM	CS
StealLink [14]	P_{TP}	100%	100%	100%
	P_{TN}	0%	0%	0%
PANDA	P_{TP}	97.6%	97.6%	99.2%
	P_{TN}	99.7%	92.8%	100%

remove graph components from the GNNs due to the node dependency in the graph [6].

E.3 Verification Efficiency of PANDA on CiteSeer and CS Datasets

We evaluate the efficiency of PANDA on the CiteSeer and CS datasets, and report the results in Table 11. Overall, PANDA continues to demonstrate high efficiency in verification on both datasets. Specifically, for the CiteSeer dataset, PANDA achieves the verification time as low as 0.04-0.08 seconds even at high verification guarantees (e.g. $\alpha = 95\%$). Compared to local verification, PANDA delivers a remarkable speedup, with verification being up to 309x faster. Even when considering the total preparation time (training the surrogate model and generating perturbations), PANDA is still able to achieve up to a 7.8x speed up over local verification. On the CS dataset, while the verification speedup is slightly less pronounced, PANDA still delivers impressive results, with the verification time up to 32x faster than local verification.

E.4 Impact of Number of Token Nodes on Performance of PANDA

For this set of experiments, we consider $\alpha, \beta = 0.9$. Following our theoretical analysis (Eqn. (3)), the required 3 token nodes for the three datasets. We vary the number of token nodes from 2 to 6 in our experiments, aiming to address the following questions: (1) Can fewer than 3 token nodes still achieve the required (α, β)-completeness, and if not, what is the extent of the shortfall? and (2) Can more than 3 token nodes guarantee (α, β)-completeness?

As shown in Table 12, PANDA achieves (α, β)-completeness only when the number of token nodes is within the specified range. Outside this range, PANDA cannot guarantee the required performance. For instance, on the LastFM dataset, PANDA can only achieve up to 88.8% P_{TN} with 6 token nodes, which is outside the range [3, 5]. This observation supports the validity of our theoretical analysis.

E.5 Impact of GCN Complexities on Verification Performance on LastFM and CS Datasets

We evaluated the performance of PANDA on more complex GCNs and multiclass classification tasks using the LastFM and CS datasets. As shown in Table 13, the results are consistent with those observed for the CiteSeer dataset. For 1-layer GCNs in binary classification tasks, PANDA consistently achieves 100% P_{TP} and P_{TN} on both datasets, supporting our theoretical results. When extending to 2-layer GCNs, there is a notable performance decrease on LastFM; however, PANDA still maintains a minimum of 97.2% for P_{TP} and 92.8% for P_{TN}. On the CS dataset, PANDA continues to perform robustly, achieving at least 99% P_{TP} and 100% P_{TN}. For 3-layer GCN, PANDA maintains a P_{TP} of at least 96.8%. However, there is a significant drop in P_{TN} on the LastFM dataset due to the increased complexity of deeper GCNs, resulting in a minimum performance of 70.8% for P_{TN}. In contrast, the CS dataset shows better resilience to higher GCN complexity, maintaining a perfect P_{TN} and a P_{TP} above 90%.

E.6 Transferrability of PANDA

In this part of the experiments, we measure the performance of PANDA when it is transferred across different GNN models. We note that PANDA does not need to transfer across datasets, as the verifier has access to the original graph. Therefore, we only consider model transferability, i.e., the target GNN model of various architectures, as well as the surrogate model used by PANDA has a different architecture from the GNNs used by the server.

Across different target models. We evaluate the performance of PANDA across different target GNN models, with GCN as the surrogate model. We consider three mainstream GNN models: GAT [31], GraphSAGE [13], and GIN [43].

Table 14 (a) reports the results, with GCN used as the surrogate model. Compared to using GCN as the target model, there is a slight decrease in both P_{TP} and P_{TN} when the target models are GAT, GraphSAGE, and GIN. Nevertheless, PANDA still maintains highly effective, with P_{TP} and P_{TN} above 92% and 91.2%, respectively. This demonstrates the transferability of PANDA across different GNN models.

Across different surrogate models. To evaluate the impact of the surrogate model on PANDA's performance, we fix the target model as a 2-layer GCN with 16 neurons per layer, and vary the architecture of the surrogate models. Since $\mathcal{V}$ (e.g., the client) may have limited resources for training the surrogate model, we consider the GCNs of simpler complexity than the target model as surrogate models. In particular, we consider four surrogate models of various architectures: (1) A simplified 2-layer GCN using a linear activation function [51], (2) a 2-layer GCN with 4 neurons per layer (i.e., $2L{\times}4N$), (3) (2) a 2-layer GCN with 8 neurons per layer (i.e., $2L{\times}8N$), and (4) a 2-layer GCN with 16 neurons per layer (i.e., the same architecture as the target model) but it is trained on a subgraph randomly sampled from the training graph whose size is 50% of the training graph.

Table 14 (b) presents the P_{TP} and P_{TN} results under different surrogate models. PANDA maintains its effectiveness for these settings, with P_{TP} and P_{TN} at least 84.4% and 72%, respectively, even when the surrogate model is a simplified GCN. Furthermore, PANDA is able to deliver P_{TP} and P_{TN} as high as 88.8% and 86%, respectively, when the surrogate model shares the same architecture as the target model but was trained on a smaller dataset. These results highlight PANDA's ability to transfer effectively across cheaper surrogate models, making it well-suited for deployment in resource-constrained verification settings.

E.7 Robustness of PANDA against Adversarial Detection

Robustness against LP detection. As the LP detection method leverages a score to detect adversarial edges, we first compare the average LP score of challenge edges and real edges, reporting the results in Table 15 (column "Score" under "LP"). The LP score reflects how likely an edge is considered malicious, with lower

Table 11: Verification efficiency (in seconds), β = 90%. We measure the time of training the surrogate model, generate perturbations, and verification, respectively, by PANDA, and compare them with the time of local verification by obtaining the unlearning model.

Required guarantee	CiteSeer				CS			
	PANDA			Local ver.	PANDA			Local ver.
α	Train surro. model	Perturbation	Ver.		Train surro. model	Perturbation	Ver.	
70%	0.729	1.31	0.04	8.406	4.94	20.28	1.01	32.17
80%		1.41	0.07	15.95		19.66	1.00	32.68
90%		1.98	0.07	21.68		23.57	1.51	49.29
95%		1.92	0.08	21.07		23.04	2.31	56.06

Table 12: Impact of number of token nodes on performance of PANDA, α, β = 90%.

Dataset	Meas.	Number of token nodes (m)				
		2	3	4	5	6
CiteSeer	P_{TP}	95.2%	98.4%	99.6%	100%	100%
	P_{TN}	100%	100%	99.6%	99.6%	99.6%
LastFM	P_{TP}	84.8%	94.0%	96.4%	98.4%	100%
	P_{TN}	96.0%	94.8%	91.4%	90.4%	88.8%
CS	P_{TP}	89.2%	99.2%	100%	100%	100%
	P_{TN}	100%	100%	100%	100%	100%

Table 13: Impact of GCN complexity of PANDA on LastFM and CS datasets.

Dataset	Meas.	1-layer GCN		2-layer GCN		3-layer GCN	
		Binary	Multi	Binary	Multi	Binary	Multi
LastFM	P_{TP}	100%	100%	97.6%	97.2%	97.6%	96.8%
	P_{TN}	100%	99.6%	95.2%	92.8%	70.8%	71.6%
CS	P_{TP}	100%	100%	100%	99.0%	90.4%	95.6%
	P_{TN}	100%	100%	100%	100%	100%	100%

Table 14: Transferability of PANDA across different types of models.

Target model	P_{TP}	P_{TN}
GCN	97.6%	100%
GAT	95.6%	95.2%
GIN	94.8%	94.8%
GraphSAGE	92.0%	93.2%

(a) Different target models (GCN as surrogate model)

Surr. model	P_{TP}	P_{TN}
Simp. GCN	84.4%	72.0%
2L×4N GCN	87.2%	79.6%
2L×8N GCN	86.6%	82.4%
2L×16N GCN	88.8%	86.0%

(b) Different surrogate models (GCN as target model)

Table 15: Comparison of characteristics of challenge edges and real edges (CiteSeer dataset).

Type	LP		OD		GGD	
	Score	Node sim.	Score	Feat. sim.	Score	Degree
Challenge edges	0.477	0.943	6.291	0.789	0.481	2.856
Real edges	0.321	0.953	21.50		0.258	1.736

values corresponding to higher maliciousness. Our results show that the average score is noticeably higher than that of real ones. To further analyze the underlying reasons, we also measure cosine similarity for both challenge edges and real ones, and report the results in Table 15 (column "Node sim." under "LP"). We observe that the cosine similarity of challenge edges is very close to that of real edges. This explains why the adversarial edges generated by PANDA are robust against LP detection.

Robustness against OD detection. Unlike the LP, the OD score represents the distance of an edge to the separating hyperplane of the OD detector, with higher values indicating a greater likelihood of being an outlier, in our case a challenge edge. We first compare the average OD score of challenge edges to that of the real edges, as reported in Table 15 (column "score" under "OD"). Our results show that the challenge edges have a significantly lower average OD score than real edges. To further investigate this, we analyze the similarity between challenge edges and real ones based on their edge features computed by the OD algorithm.

Table 15 (column "Feat. sim." under "OD") reports the average pairwise cosine similarity of OD edge features for challenge edges and real ones. The results indicate that the OD edge features of the challenge and real edges have the same average cosine similarity. In other words, challenge edges resemble real ones in the space of the OD features, making them difficult for OD to detect.

Robustness against GGD detection. Recall that GGD also leverages a score to detect the adversarial edges. The score indicates the likelihood that an edge is generated by the generative model of GGD. A lower score indicates that the edge is more likely to be adversarial. Thus we compare the average GGD score of challenge edges and real ones. Table 15 (Column "Score" under "GGD") presents the 25th percentile GGD score for both edge types. We observe that challenge edges receive a significantly higher GGD score than real ones. Since a higher GGD score indicates a lower likelihood of an edge being adversarial, this explains why GGD fails to detect the adversarial edges. We perform some additional analysis to explore the reason behind this observation. As GGD randomly samples the subgraphs from the given graph, and considers the edges included in these subgraphs less likely to be malicious, the edges associated with high-degree nodes are more likely to be sampled and are less likely to be malicious. Following this reasoning, we measure the average node degree of both real and challenge edges, and report them in Table 15 (Column "Degree" under "GGD"). We observe that the nodes associated with challenge edges have a higher degree than real ones. This explains the low detection accuracy of GGD, as the challenge edges are more likely to be sampled by GGD than real ones.